Dynamic Modeling and Control of a Car-Like Robot

Eric N Moret

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Electrical Engineering

February 5, 2003
Blacksburg, Virginia

Keywords: autonomous vehicle, intelligent transportation system, dynamic, nonlinear control, nonholonomic

Virginia Polytechnic Institute
and State University

Dynamic Modeling and Control of a Car-Like Robot

Eric N Moret

(ABSTRACT)

The Flexible Low-cost Automated Scaled Highway (FLASH) laboratory at the Virginia Tech Transportation Institute (VTTI) is one of many facilities dedicated to the field of Intelligent Transportation Systems (ITS). The goal of the FLASH lab is to provide small-scale development and implementation of autonomous control strategies for today's vehicles.

The current controller used on the scale vehicles is based solely on the kinematics of the system. This body of work was aimed to develop a dynamic control law to enhance the performance of the existing kinematic controller. This control system is intended to automatically maintain the vehicle's alignment on the road as well as keep the speed of the vehicle constant. Implementation of such systems could conceivably reduce driver fatigue by removing nearly all the burden of the driving process from the driver while on the highway.

System dynamics of car-like robots with nonholonomic constraints were employed in this research to create a controller for an autonomous path following vehicle. The application of working kinematic and dynamic models describing car-like robotic systems allowed the development of a nonlinear controller.

Simulations of the vehicle and controller were done using MATLAB. Comparisons of the kinematic controller and the dynamic controller presented here were also done. In order to make the simulations model the actual system more closely, measures were taken to approximate actual sensor readings.

Acknowledgments

The author wishes to thank Dr. Pushkin Kachroo for his support and advise, which without, this thesis would not be possible. His endless enthusiasm and patience is something to be admired and sought after in both academic settings as well as life.

To Dr. Harley Cudney, thanks must be given to the man who introduced the author to his passion.

The author wishes to thank Patricia Mellodge for her patience and support. Her work has been, and will continue to be, the building blocks of the FLASH project.

Thanks go to Richard Henry, for always having caps to eat.

This work is dedicated to my best friend, inspiration, and life love. Brighid, you are the reason I strive for excellence. I would be lost without you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In 1769, the first self-propelled vehicle was created. The inventor, Nicolas Joseph Cugnot, used a steam engine for power, which propelled him at an incredible speed of 2 1/2 miles per hour. In 1771, the first automotive accident was recorded. Mr. Cugnot had driven his vehicle into a stone wall [10].

With the invention of the automobile, came the inevitability of automotive accidents and injuries. Since that time, there have been many advances in the automotive industry to improve speed and efficiency. For example, the first gasoline-powered vehicle appeared in 1864, traveling at a speed of 10 mph [10]. The initial concern of the automotive companies at the time was not safety, but building a better, more powerful vehicle. Of course, as the vehicles became faster, they also became more harmful. This is what sparked the creation of safety features such as hydraulic brakes, which appeared in 1920, and safety glass, making its debut in 1926 [11][10]. Finally, almost 150 years after the creation of the first automobile, safety started taking a larger role in the area of automobiles. Although Volvo had introduced safety belts as early as 1849, it wasn't until 1930 that American physicians began urging automobile manufacturers to install belts as standard equipment. The first American car company to issue seat belts as standard equipment was Studebaker-Packard in 1964. Finally, in 1973, air bags were first introduced as the next step in vehicle safety [10].

As the safety of vehicles caught up, many turned their attention to making the driving experience more comfortable. The first of these comfort extras was the car radio, invented in 1929 by a small company called Motorola. The next major step was the introduction of an automatic transmission created by General Motors in 1939. The following year brought the invention of the first automotive air conditioner, courtesy of Packard. In 1945, an engineer by the name of Teetor, inspired by the poor driving habits of his lawyer friend, invented cruise control [10].

1.1 Motivation

With the invention of cruise control, the burden of driving was decreased for anyone driving on today's interstate system. As time moved forward; power steering, anti-lock braking, and traction control were created to further alleviate stress from the driver. This leads to the driver being able to give more attention to other tasks. It only makes sense that the next step is to completely automate the driving experience.

There are many advantages for automating vehicles. The first is to lessen the burden of the drive from the driver. With families being more spread out, the automation would lead to less time spent between destinations by decreasing the number of stops to allow the driver to rest. Further, the driver would be more rested when reaching their destination. Both of these would lead to more time available to be spent with family and friends.

The second, and more important, reason for automating the driving process is safety. In 2001, there were approximately 6,393,000 automotive accidents, leading to over 41,000 deaths and more than 3,000,000 injured [1]. The causes of these accidents come from a wide array of backgrounds, but all lead to being error on the part of the driver. If this error were removed, even in the highway setting, then the number of accidents each year would surely decrease.

Finally, the majority of control algorithms available for such a vehicle only use the kinematic model, such as [2]. The advantage of the kinematic model is that it keeps the steering and velocity of the vehicle completely decoupled. This means that the control of the two is completely separate. The problem with this is that, in the process, the dynamics of the vehicle are ignored.

As will be shown, the assumption that the steering can be controlled without taking velocity dynamics into account is valid, but the opposite is not true. The velocity of a standard, front-steer, rear-drive vehicle is very dependant upon the dynamics of the steering. As a result, in order to control the speed of the vehicle, one must understand the dynamics of the vehicle, as well as dynamics of the steering and velocity actuators. While the control algorithm presented here is specifically designed for use on the 1/10th scale model car being employed on the Virginia Tech Flexible Low-cost Automated Scaled Highway (FLASH) project, it is applicable to any general car-like robot.

1.2 FLASH Laboratory

The FLASH Lab has two primary goals. The first of these goals is to create an educational display at the Science Museum of Virginia, located in Richmond, and the Virginia Museum of Transportation located in Roanoke. These exhibits will have two major topics. The first is to increase awareness of the existent safety technologies currently used in today's vehicles. Some of the technologies to be included are:

- Safety belts
- Air bags
- Safety glass in windshields
- Anti-lock braking systems
- Advancements in tire manufacturing

Generally, people are hesitant to put faith in new technologies, especially when those technologies threaten to take control of the activities that people view as being part of their everyday lives. For this reason, these exhibits will also introduce the public to the possible future of driving on the interstate system. It is the goal of this exhibit to prepare and inform the public so when these technologies become available, they are not taken aback. Some of these technologies are:

- Infrared sensors for lateral control
- Magnetic sensors for lateral control
- Camera and image processing for lateral control
- Ultrasonic sensors for adaptive cruise control

All of this information will be available in the form of interactive displays. Each display will have both an overview, as well as in-depth information into a specific technology. To further enhance the audience's understanding of the future technologies, several autonomous 1/10 scale vehicles will be traveling around an enclosed track. The current concept design for this track is shown in Figure 1.1.

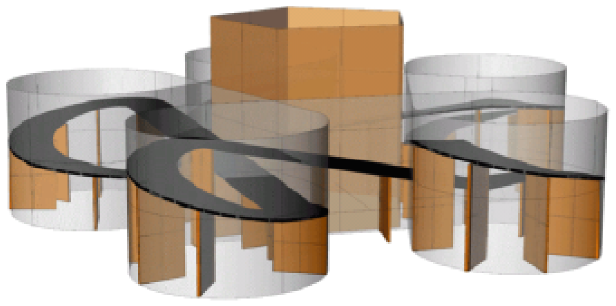

Figure 1.1: Current concept design of *FLASH* museum track [8]

These autonomous vehicles are the main focus of the engineering group within the FLASH Lab. The second goal of the FLASH project is to serve as a small-scale test bed for ITS.

1.3 Previous Research

The mission of ITS is to make the driving experience completely automated. With a charge as ambiguous as this, the focus in automation has been spread in many different directions. There are those who focus on control strategies in adaptive cruise control [7], whose focus is to slow a vehicle as it approaches a slower-moving vehicle ahead.

Another direction that has been followed is that in parameter estimation. In order to make control strategies available to vehicles already in the road, estimation must be made in place of direct sensing. The specific control strategies lend themselves to different needs in terms of which parameters must be known. For the controller derived in [18], it is necessary to estimate the steering angle of the vehicle. The main focus of [6] is how one might estimate the curvature of the road. This controller is based on the chained system derived in [2], which will be looked into more depth later in this work.

Traffic avoidance and mapped path tracking is the topic of [24]. Garnier, et al, merge a map-based path following system with that of a collision avoidance control into a hybrid fuzzy-logic controller. As the vehicle navigates its way through the mapped path, it allows for unexpected road-blocks and devises escape paths as necessary. What this controller does not focus on is how the vehicle is controlled to travel along the path given.

The largest body of work devoted to ITS is that in the direction of vehicular control for point stabilization and path tracking.

1.3.1 Point Stabilization

The problem of point stabilization for today's car arises because of the nonholonomic constraints created with the design of a standard vehicle. The easiest example of these constraints is in the problem of parallel parking. While a car cannot move directly into the parking spot, it can reach that point. While it is a requirement to receive ones driver's license, it poses an interesting complication to automation. As a result, many strategies have been developed.

One strategy employs state-space linearization [13]. Linearization is not, generally, a possibility to stabilization of a vehicle, due to the nonholonomic constraints mentioned above. The authors in [13] have proposed a new set of coordinates which lends itself to control via well-established linear time-invariant strategies.

The majority of the controllers available today, especially for point stabilization, are designed around nonlinear control schemes. In [14], a controller is derived, which is actually a combination of two piecewise continuous controllers. The first controller derived is one that drives the orientation of the vehicle to a specified angle. The second controller is used when the vehicle is more than a specified distance from its target. Once the vehicle is "close enough", then the first controller takes over.

One of the most used nonlinear control strategies is that of sliding-mode. It is this control strategy which is employed in [12]. The sliding-mode controller is applied to a three-wheeled vehicle, whose third wheel is a caster. As a result, the control strategy differs from that of a standard automobile. Moreover, the fact that the third wheel is a castor allows the vehicle to pivot directly about the center of its rear axle, which alleviates some of the issues related to the nonholonomic constraints associated with today's vehicles.

In 1982, a mostly forgotten design strategy for control was brought back into academic interest. This strategy is known as neural networks. It is this strategy that is employed in [23]. The controller designed in the work implements its neural network on another three-wheeled vehicle. While this type of vehicle is not the focus of this thesis, the reference is included to show the range of control strategies used to solve the problems of point stabilization.

As time has progressed, new control strategies have been developed. One of these new strategies is known as hybrid control. The key behind hybrid control is the ability to switch between continuous control strategies, based on predefined metrics. Control strategies for point stabilization using hybrid control have also been developed [25].

1.3.2 Path Tracking

Path tracking is defined as the ability for an object to travel along a prescribed path, without a prescribed velocity. This is the focus of the following papers, as well as that of this thesis.

Linearization is almost always considered when creating controllers for nonlinear systems. The same is true in this respect, and this is the control design approached in [16]. Shih, et al, use proven linearization techniques in order to navigate a predefined path.

The control strategy presented in [17] is derived from a nonlinear approach. This controller is designed to be general, and is used to control both a two-wheeled vehicle, as well as a normal front-wheel drive vehicle. This controller was also designed to navigate a predefined path.

Neural networks and fuzzy logic as also been employed in the path-tracking problem [21],[22]. A predefined path is required for [21], while [22] can be set on a path, which it will then navigate.

All the control strategies mentioned above have been based on kinematic models. In reference to point stabilization, using the kinematic models alone is reasonable because the speeds being dealt with are very slow. While the kinematic model works for path tracking, a dynamic model can operate more efficiently, with less error. One such controller is designed in [20]. This controller takes the dynamics of the vehicle into account, yet still assumes a constant velocity, and uses a predefined path.

Another controller, derived in [19], employs the dynamics of the vehicle, yet does not take into account the dynamics of the actuators. It also assumes that the path is predefined.

All of the systems noted above have been based around the nonholonomic constraints. These constraints are dependant upon the assumption that there is zero slip. This is not the case for controllers such as [5]. This controller is designed to, not only control a vehicle dynamically, but takes wheel slip and other losses into account.

One thing all the above systems have in common is that they are not chained systems. The advantage to chained systems is that they allow for multiple trailers to be attached to the driving vehicle, yet does not augment the control law. Such controllers have been designed in [15] and [2]. While these systems do employ chained form, they are both strictly kinematic controllers, and do not take the dynamics of the system into account.

1.4 Contributions To and From This Thesis

From all the research done, there was one control strategy that was not found. The advantages of chained form are obvious in the fact that many vehicles on the highway today carry, or have the ability to carry, multiple trailers. Furthermore, including the dynamics of the system in the control design has been shown to improve response in innumerable applications. Finally, in keeping with the scope of the FLASH project, it is known, apriori, that the turning radii are large, leading to small changes in the steering angle. With that in mind, the assumption of nonholonomic constraints can be used.

The result: this control law designed herein takes into account the dynamics of the vehicle, with contributions from [19] in its derivation, as well as the dynamics of the actuators. The actuators involved are a servo, whose response was found experimentally, and an electric motor, whose model was defined as found in [4]. Finally, this dynamically derived model was integrated into the chained form currently being used on the FLASH project [2], and backstepping, as derived in [3], was used to complete the controller.

Chapter 2

Mathematical Model

This chapter discusses the mathematical model used for the vehicle. It has been separated into two main sections: Kinematic and Dynamic. The model derived here is what the controller, described in the following chapter, is based upon.

2.1 Derivation of the Kinematic Model

The first step taken in the derivation is to create the kinematic model by employing the nonholonomic constraints. These constraints hold under the assumption that there is no slippage at the wheel. A nonholonomic constraint is one that is not integrable. The constraints related to an automobile are those of the vehicle's velocity. As a result, the general form of the nonholonomic constraint is

$$\dot{u}\sin(\theta) - \dot{w}\cos(\theta) = 0 \tag{2.1}$$

Where $\dot{u}$ and $\dot{w}$ are the velocities of a wheel within a given (u, w) coordinate system, and θ is the angle of the wheel with respect to the x-axis. With this in mind, we examine a general front-wheel steer, rear-wheel drive vehicle. For small angles of steering, the car can be modeled as a bicycle, as shown in Figure 2.1.

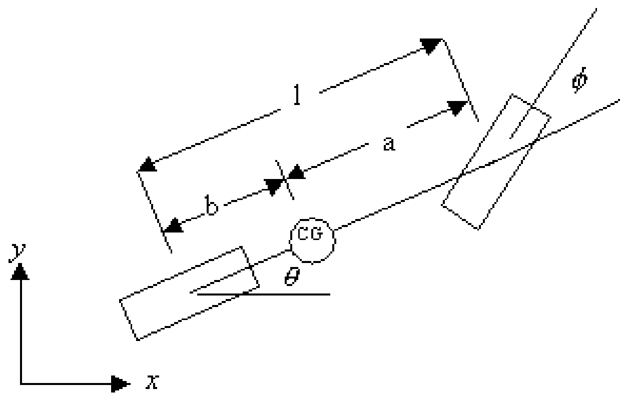

Figure 2.1: General Coordinates for a car-like robot

Denote (x, y) as being the position of the center of gravity, θ as the orientation of the vehicle with respect to the x-axis, and ϕ as the steering angle between the front wheel and the body axis. Let (x_1, y_1) denote the position of the rear axle, and (x_2, y_2) denote the position of the front axle. This defines,

$$\begin{array}{ll} x_1 = x - b\cos(\theta) & x_2 = x + a\cos(\theta) \\ y_1 = y - b\sin(\theta) & y_2 = y + a\sin(\theta) \end{array}$$

$$\begin{array}{ll} \dot{x_1} = \dot{x} + b\dot{\theta}\sin(\theta) & \dot{x_2} = \dot{x} - a\dot{\theta}\sin(\theta) \\ \dot{y_1} = \dot{y} - b\dot{\theta}\cos(\theta) & \dot{y_2} = \dot{y} + a\dot{\theta}\cos(\theta) \end{array}$$

The nonholonomic constraints are then written for each wheel, resulting in

$$\dot{x_1}\sin(\theta) - \dot{y_1}\cos(\theta) = 0 \tag{2.2}$$

$$\dot{x_2}\sin(\theta+\phi) - \dot{y_2}\cos(\theta+\phi) = 0 \tag{2.3}$$

Substituting the definition of $(\dot{x_1}, \dot{y_1})$ and $(\dot{x_2}, \dot{y_2})$ into equations 2.2 and 2.3 gives

$$\dot{x}\sin(\theta) - \dot{y}\cos(\theta) + b\dot{\theta} = 0 \tag{2.4}$$

$$\dot{x}\sin(\theta+\phi) - \dot{y}\cos(\theta+\phi) + a\dot{\theta}\cos(\theta) = 0 \tag{2.5}$$

Using the coordinate frame of the vehicle, as shown in Figure 2.2, define the u-axis as being that which is along the length of the vehicle and the w-axis normal to the u-axis,

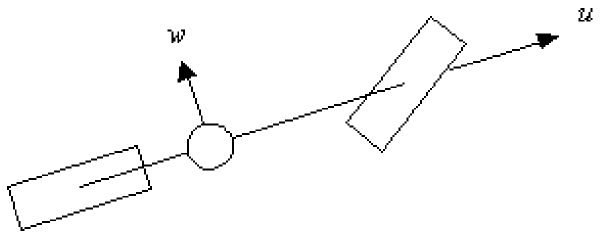

Figure 2.2: Car body coordinates

Define

$$\dot{x} = v_u \cos(\theta) - v_w \sin(\theta)$$

$$\dot{y} = v_u \sin(\theta) + v_w \cos(\theta)$$

Where v_u and v_w are the velocities of the center of gravity along the u and w axes, respectively. Substituting these definitions into equations 2.4 and 2.5 results with

$$v_w = \dot{\theta} b \tag{2.6}$$

$$\dot{\theta} = \frac{\tan\phi}{l} v_u \tag{2.7}$$

Thus, the derivatives of the nonholonomic equations are

$$\dot{v_w} = \ddot{\theta} b \tag{2.8}$$

$$\ddot{\theta} = \frac{\tan\phi}{l} \dot{v_u} + \frac{v_u}{l(\cos\phi)^2} \dot{\phi} \tag{2.9}$$

2.2 Derivation of the Dynamic Model

Now that the kinematic constraints are in a more useful format, the dynamic equations can be derived. These dynamic equations are similar to those found in [19], with a couple added assumptions. These added assumptions are that there is no friction force between the wheels and the vehicle, and that the rear wheels are locked to be in the same orientation as the vehicle. Other assumptions, used both in [19] and this paper is that there is no slip at the wheel, and that the driving force, based on the radius of the wheel and the drive torque, can be modeled as acting at the center of the rear wheels. With the slippage assumption comes a pair of forces, one acting at each wheel, perpendicular to that wheel. The forces involved in this derivation are shown in Figure 2.3.

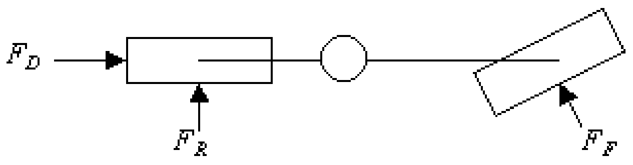

Figure 2.3: Car Input and resultant forces

The resultant dynamic equations are

$$\dot{v_u} = v_w\dot{\theta} - \frac{F_F \sin\phi}{m} + \frac{F_D}{m} \tag{2.10}$$

$$\dot{v_w} = -v_u\dot{\theta} + \frac{F_F \cos\phi}{m} + \frac{F_R}{m} \tag{2.11}$$

$$\ddot{\theta} = \frac{aF_F \cos\phi}{J} - \frac{bF_R}{J} \tag{2.12}$$

Where m and J are the mass and mass moment of inertia of the vehicle about the center of gravity. F_D is the driving force, applied at the rear axle, along the u-axis, and F_F and F_R are the resultant lateral forces on the front and rear tires respectively.

Solving equation 2.12 in terms of F_R gives

$$F_R = \frac{aF_F \cos\phi}{b} - \frac{J\ddot{\theta}}{b} \tag{2.13}$$

Substituting equations 2.8 and 2.13 into equation 2.11 reveals

$$\ddot{\theta}b = -v_u\dot{\theta} + \frac{F_F \cos\phi}{m} + \frac{aF_F \cos\phi}{bm} - \frac{J\ddot{\theta}}{bm} \tag{2.14}$$

Solving equation 2.14 or F_F gives

$$F_F = \frac{b^2m + J}{l\cos\phi}\ddot{\theta} + \frac{bm}{l\cos\phi}v_u\dot{\theta} \tag{2.15}$$

Placing equations 2.6, 2.7, 2.9, and 2.15 into equation 2.10 gives

$$\dot{v_u} = \frac{v_u(b^2m + J)\tan\phi}{\gamma}\dot{\phi} + \frac{l^2(\cos\phi)^2}{\gamma}F_D \tag{2.16}$$

$$\gamma = (\cos\phi)^2[l^2m + (b^2m + J)(\tan\phi)^2] \tag{2.17}$$

Placing equations 2.6 and 2.7 into the definitions of velocity and choosing states to be $X = [x \quad y \quad \theta \quad v_u \quad F_D \quad \phi]'$, finds

$$\begin{aligned}
\dot{x} &= \left[\cos\theta - \frac{b\tan\phi}{l}\sin\theta\right] v_u \\
\dot{y} &= \left[\sin\theta + \frac{b\tan\phi}{l}\cos\theta\right] v_u \\
\dot{\theta} &= \frac{\tan\phi}{l} v_u \\
\dot{v_u} &= \frac{v_u(b^2m+J)\tan\phi}{\gamma}\dot{\phi} + \frac{l^2(\cos\phi)^2}{\gamma}F_D \\
\dot{F_D} &= f\left(F_D, v_u, u_1\right) \\
\dot{\phi} &= f\left(\phi, u_2\right)
\end{aligned} \tag{2.18}$$

Where γ is defined by equation 2.17 and u_1 and u_2 are the input voltages, ranging from -5V to 5V. The final step in deriving the dynamics of the system is to find equations for the driving force and the steering servo. Since it is known that the driving force is transmitted to the vehicle through an armature-controlled DC motor, the assumed model is derived as follows.

$$T_m = K_m I_a \quad , \quad I_a = \frac{u_1 - K_b\omega}{R_a + L_a s}$$
$$T_m = J_m\dot{\omega} + b_m\omega + T_D$$

$$K_m\frac{u_1 - K_b\omega}{R_a + L_a s} = J_m\dot{\omega} + b_m\omega + T_D \tag{2.19}$$

Where $K_m, K_b, R_b, L_a, J_m, and b_m$ are all motor constants, ω is the angular velocity of the motor, and T_D is the driving torque. Solving equation 2.19 for the driving torque and ignoring the higher order terms, which are three orders of magnitude less or smaller, yields

$$\dot{T_D} = -\frac{R_a}{L_a}T_D - \frac{(K_mK_b + R_ab_m)}{L_a}\omega + \frac{K_m}{L_a}u_1 \tag{2.20}$$

Solving T_D and ω in terms of F_D and v_u gives

$$T_D = \frac{N_mR_w}{N_w}F_D \quad , \quad \omega = \frac{N_w}{N_mR_w}v_u$$

$$\dot{F_D} = -\frac{R_a}{L_a}F_D - \frac{(K_m K_b + R_a b_m)\,N_w^2}{L_a N_m^2 R_w^2}v_u + \frac{K_m N_w}{L_a N_m R_w}u_1 \tag{2.21}$$

Where R_w is the radius of the wheel and N_w and N_m are the number if teeth on the gears connecting the axle and motor respectively. The servo was assumed, and experimentally proven, to be well-represented by a linear first order system of the form

$$\dot{\phi} = \frac{1}{\tau_s}\phi + c_s u_2 \tag{2.22}$$

With equations 2.21 and 2.22, the full state equations can be re-written as

$$\begin{aligned}
\dot{x} &= \left[\cos\theta - \frac{b\tan\phi}{l}\sin\theta\right]v_u \\
\dot{y} &= \left[\sin\theta + \frac{b\tan\phi}{l}\cos\theta\right]v_u \\
\dot{\theta} &= \frac{\tan\phi}{l}v_u \\
\dot{v_u} &= \frac{v_u(b^2 m + J)\tan\phi}{\gamma}\dot{\phi} + \frac{l^2(\cos\phi)^2}{\gamma}F_D \\
\dot{F_D} &= -\frac{R_a}{L_a}F_D - \frac{(K_m K_b + R_a b_m)\,N_w^2}{L_a N_m^2 R_w^2}v_u + \frac{K_m N_w}{L_a N_m R_w}u_1 \\
\dot{\phi} &= \frac{1}{\tau_s}\phi + c_s u_2
\end{aligned} \tag{2.23}$$

with γ still being defined as in equation 2.17.

Chapter 3

Control Algorithms

Now that the complete mathematical model has been identified, the controllers can be designed. It is not the intent of the author to completely redesign the controller used on the FLASH project. Instead, the intent is to append the current controller to take the dynamics of the system into account. Since the steering component of the model is not dependant upon the velocity, it will be discussed first.

3.1 Design of the Lateral Controller

The goal of the lateral controller was to put $\dot{\phi}$ in the form that appears in [2]. This form is

$$\dot{\phi} = v_2 \tag{3.1}$$

With this in mind, the first transform of variables becomes

$$u_2 = \frac{v_2 - \frac{1}{\tau_s}\phi}{c_s} \tag{3.2}$$

From here, the control law from [2] is looked at. The first step in the controller is to convert the kinematic states from global coordinates to path coordinates. Since the velocity, force, and steering equations are not dependant upon the x and y position of the vehicle, those coordinates can be redefined as being the rear axle of the vehicle. As a result, b in the first two state equations becomes zero. Ignoring the velocity and force equations, the remaining states are

$$\left\{\begin{array}{c}\dot{x}\\ \dot{y}\\ \dot{\theta}\\ \dot{\phi}\end{array}\right\} = \left[\begin{array}{c}\cos\theta\\ \sin\theta\\ \frac{\tan\phi}{l}\\ 0\end{array}\right] v_u + \left[\begin{array}{c}0\\0\\0\\1\end{array}\right] v_2 \tag{3.3}$$

Next, the states are re-written in terms of the path coordinates

$$\left\{\begin{array}{c}\dot{s}\\ \dot{d}\\ \dot{\theta}_p\\ \dot{\phi}\end{array}\right\} = \left[\begin{array}{c}\frac{\cos\theta_p}{1-dc(s)}\\ \sin\theta_p\\ \left(\frac{\tan\phi}{l} - \frac{c(s)\cos\theta_p}{1-dc(s)}\right)\\ 0\end{array}\right] v_u + \left[\begin{array}{c}0\\0\\0\\1\end{array}\right] v_2 \tag{3.4}$$

Where $c(s)$ is the curvature of the path, and s and d are defined as shown in Figure 3.1

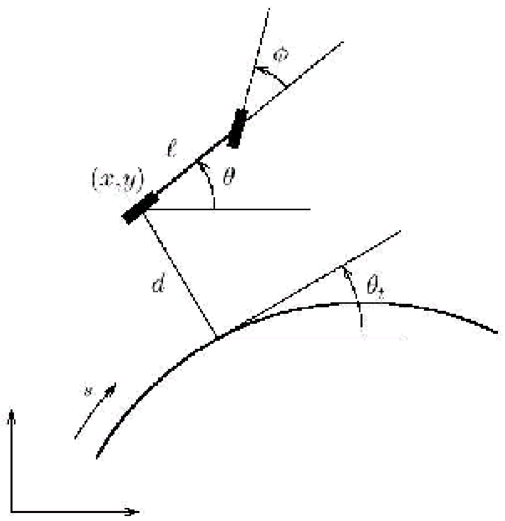

Figure 3.1: Coordinate definition for a path following task

and θ_p is defined as $\theta - \theta_t$. The next step is to convert these coordinates into $(2,4)$ chained form. The general form for a $(2,4)$ chained system, with slight modification to avoid redundant variables, is

$$\begin{aligned}\dot{x_1} &= v_s\\ \dot{x_2} &= w_2\\ \dot{x_3} &= x_2 v_s\\ \dot{x_4} &= x_3 v_s\end{aligned}$$

In order to do this, the following change of coordinates are made:

$$\begin{aligned} x_1 &= s \\ x_2 &= -c'(s)d\tan\theta_p - c(s)\left(1-dc(s)\right)\frac{1+\left(\sin\theta_p\right)^2}{\left(\cos\theta_p\right)^2} + \frac{\left(1-dc(s)\right)^2\tan\phi}{l\left(\cos\theta_p\right)^3} \\ x_3 &= \left(1-dc(s)\right)\tan\theta_p \\ x_4 &= d \end{aligned} \tag{3.5}$$

Where $c'(s)$ is defined s the derivative of the curvature with respect to s. The inputs are then transformed into

$$\begin{aligned} v_u &= \frac{1-dc(s)}{\cos\theta_p}v_s \\ v_2 &= \alpha_2(w_2 - \alpha_1 v_s) \end{aligned} \tag{3.6}$$

Where α_1 and α_2 are defined as

$$\begin{aligned} \alpha_1 &= \frac{\delta x_2}{\delta s} + \frac{\delta x_2}{\delta d}\left(1-dc(s)\right)\tan\theta_p + \frac{\delta x_2}{\delta\theta_p}\left(\frac{\tan\phi\left(1-dc(s)\right)}{l\cos\theta_p} - c(s)\right) \\ \alpha_2 &= \frac{l\left(\cos\theta_p\right)^3\left(\cos\phi\right)^2}{\left(1-dc(s)\right)^2} \end{aligned} \tag{3.7}$$

Next, the states are rearranged into

$$\begin{aligned} \chi_1 &= x_1 \\ \chi_2 &= x_4 \\ \chi_3 &= x_3 \\ \chi_4 &= x_2 \end{aligned}$$

And, finally, the control law is defined as

$$w_2 = -k_1|v_s|\chi_2 - k_2 v_s\chi_3 - k_3|v_s|\chi_4 \tag{3.8}$$

Where k_1, k_2, and k_3 are arbitrary gains. The main difference between the control law designed here and the one designed in [2] is that this one takes the dynamics of the servo into account. As will be shown in the simulation chapter, this small addition makes for much less error.

3.2 Design of the Velocity Controllers

The major shortfall of the controller designed in [2], as well as other path-following control algorithms, is that they assume the velocity is directly controlled. Unfortunately, as the state equations 2.23 example, this is not the case for most systems. In an attempt to keep the advantages of non-linear control, feedback linearization did not seem to be a viable solution. In that light, the controller was designed using standard backstepping techniques, as laid out in [3], with one small variation.

3.2.1 Full-state Velocity Controller

First, focus on the lower three states from equations 2.23.

$$\begin{aligned}
\dot{v_u} &= \frac{v_u(b^2m+J)\tan\phi}{\gamma}\dot{\phi} + \frac{l^2(\cos\phi)^2}{\gamma}F_D \\
\dot{F_D} &= -\frac{R_a}{L_a}F_D - \frac{(K_mK_b+R_ab_m)\,N_w^2}{L_aN_m^2R_w^2}v_u + \frac{K_mN_w}{L_aN_mR_w}u_1 \\
\dot{\phi} &= \frac{1}{\tau_s}\phi + c_su_2 \\
\gamma &= (\cos\phi)^2[l^2m+(b^2m+J)(\tan\phi)^2]
\end{aligned} \tag{3.9}$$

Substituting equation 3.2 into equation 3.9, and choosing

$$u_1 = \frac{R_wN_mL_a}{N_wK_m}\left(w_1 + \frac{(R_ab_m+K_mK_b)\,N_w^2}{N_m^2R_w^2} + \frac{R_a}{L_a}F_D\right) \tag{3.10}$$

Results in

$$\begin{aligned}
\dot{v_u} &= -\frac{v_u\,(b^2m+J)\tan\phi}{\gamma}v_2 + \frac{l^2(\cos\phi)^2}{\gamma}F_D \\
\dot{F_D} &= w_1 \\
\dot{\phi} &= v_2
\end{aligned} \tag{3.11}$$

Where v_2 is defined in equation 3.6. Now, define η (Φ in [3]) as

$$\eta = \frac{\gamma}{l^2(\cos\phi)^2}\left(-k_{v1}v_u + \frac{v_u\,(\tan\phi)\,(b^2m+J)}{\gamma}v_2\right) \tag{3.12}$$

Choosing

$$V(v_u) = \frac{1}{2}v_u^2 \quad \Rightarrow \quad \dot{V} = v_u \dot{v_u} = -k_{v1}v_u^2$$

As a result, the velocity and force equations become

$$\begin{aligned}
\dot{v_u} &= \frac{v_u\left(b^2m + J\right)\tan\phi}{\gamma}v_2 + \frac{l^2\left(\cos\phi\right)^2}{\gamma}\eta + \frac{l^2\left(\cos\phi\right)^2}{\gamma}z \\
z &= F_D - \eta \\
\dot{z} &= w_1 - \dot{\eta}
\end{aligned} \tag{3.13}$$

Finally, following [3], set

$$\begin{aligned}
w_1 &= v_1 + \dot{\eta} \\
v_1 &= -\frac{\delta V}{\delta v_u}g(v_u) - k_{v2}z \quad \Rightarrow \quad v_1 = -\frac{v_u l^2\left(\cos\phi\right)^2}{\gamma} - k_{v2}\left(F_D - \eta\right)
\end{aligned} \tag{3.14}$$

The variation from [3] comes in when evaluating w_1. In [3], η is considered to be fully defined in terms of the states leading up to its definition. In this case, η is a function of all the states, including those not within the derivation of the controller. As a result, a more general form of backstepping must be used. Instead of using

$$w_1 = v_1 + \frac{\delta\eta}{\delta x}\dot{x}$$

The transform becomes

$$w_1 = v_1 + \frac{d\eta}{dt}$$

Where

$$\frac{d\eta}{dt} = \frac{d}{dt}\left[\frac{\gamma}{l^2\left(\cos\phi\right)^2}\left(-k_{v1}v_u + \frac{v_u\tan\phi\left(b^2m + J\right)}{\gamma}v_2\right)\right] \tag{3.15}$$

Due to the length of the derivative, it is not shown here, but can be found in Appendix A in the form that $MatLab^{TM}$ outputs.

The controller derived above does have one flaw. It drives the velocity to zero. Since the goal of this controller is to drive the velocity to a specific value, a slight change must be made to the states. Specifically, η is changed to

$$\eta = \frac{\gamma}{l^2(\cos\phi)^2}\left(-k_{v1}(v_u - v_d) + \dot{v_d} + \frac{v_u\,(\tan\phi)\,(b^2m+J)}{\gamma}v_2\right) \tag{3.16}$$

Where v_d, the desired velocity, and its derivative are given by the user upon startup. This leads to the following changes in states:

$$\begin{aligned}
\dot{v_u} &= \dot{v_d} - k_{v1}(v_u - v_d) \;\Rightarrow\; \dot{e_v} = -k_{v1}e_v \\
e_v &= v_u - v_d
\end{aligned}$$

Combining equations 3.4, 3.11, 3.13, and 3.16 gives the state equations as

$$\begin{aligned}
\dot{s} &= \frac{\cos\theta_p}{1-dc(s)}(e_v + v_d) \\
\dot{d} &= \sin\theta_p(e_v + v_d) \\
\dot{\theta}_p &= \left(\frac{\tan\phi}{l} - \frac{c(s)\cos\theta_p}{1-dc(s)}\right)(e_v + v_d) \\
\dot{e_v} &= -\frac{v_u\,(b^2m+J)\tan\phi}{\gamma}v_2 + \frac{l^2\,(\cos\phi)^2}{\gamma}\eta + \frac{l^2\,(\cos\phi)^2}{\gamma}z \\
\dot{z} &= v_1 \\
\dot{\phi} &= v_2
\end{aligned} \tag{3.17}$$

$$\begin{aligned}
\eta &= \frac{\gamma}{l^2(\cos\phi)^2}\left(-k_{v1}(v_u - v_d) + \dot{v_d} + \frac{v_u\,(\tan\phi)\,(b^2m+J)}{\gamma}v_2\right) \\
\gamma &= (\cos\phi)^2[l^2m + (b^2m+J)(\tan\phi)^2]
\end{aligned}$$

With the controllers being defined as

$$\begin{aligned}
v_1 &= -\frac{v_u l^2(\cos\phi)^2}{\gamma} - k_{v2}(F_D - \eta) \\
w_1 &= v_1 + \frac{d\eta}{dt} \\
u_1 &= \frac{R_w N_m L_a}{N_w K_m}\left(w_1 + \frac{(R_a b_m + K_m K_b)N_w^2}{N_m^2 R_w^2}v_u + \frac{R_a}{L_a}F_D\right)
\end{aligned}$$

$$\begin{aligned}
v_2 &= \alpha_2(w_2 - \alpha_1 v_s) \\
w_2 &= -k_1|v_s|\chi_2 - k_2 v_s \chi_3 - k_3|v_s|\chi_4 \\
\alpha_1 &= \frac{\delta x_2}{\delta s} + \frac{\delta x_2}{\delta d}\left(1 - dc(s)\right)\tan\theta_p + \frac{\delta x_2}{\delta \theta_p}\left(\frac{\tan\phi\left(1 - dc(s)\right)}{l\cos\theta_p} - c(s)\right) \\
\alpha_2 &= \frac{l\left(\cos\theta_p\right)^3\left(\cos\phi\right)^2}{\left(1 - dc(s)\right)^2} \\
v_s &= \frac{\cos\theta_p}{1 - dc(s)} v_u \\
u_2 &= \frac{v_2 - \frac{1}{\tau_s}\phi}{c_s}
\end{aligned}$$

3.2.2 Truncated Velocity Controller

In an effort to make the controller more practical from an application standpoint, a mistake made in the derivation in the beginning of this research was reintroduced. The derivation for the truncated controller is identical to the full-state controller until equation 3.14. It is at this point where, rather than using the generalized form of w_1, the form as it appears in [3] is used. The result is as follows.

The state equations involved are

$$\begin{aligned}
\dot{v_u} &= -\frac{v_u\left(b^2 m + J\right)\tan\phi}{\gamma} v_2 + \frac{l^2(\cos\phi)^2}{\gamma} F_D \\
\dot{F_D} &= w_1 \\
\dot{\phi} &= v_2
\end{aligned} \tag{3.18}$$

With η being defined as

$$\begin{aligned}
\eta &= \frac{\gamma}{l^2(\cos\phi)^2}\left(-k_{v1}(v_u - v_d) + \dot{v_d} + \frac{v_u\left(\tan\phi\right)\left(b^2 m + J\right)}{\gamma} v_2\right) \\
\gamma &= (\cos\phi)^2[l^2 m + (b^2 m + J)(\tan\phi)^2]
\end{aligned}$$

Where w_1 id defined as

$$w_1 = v_1 + \frac{\delta\eta}{\delta x}\dot{x}$$

Using this definition of w_1 produces

$$w_1 = v_1 + \frac{\gamma}{l^2(\cos\phi)^2}\left(-k_{v1} + \frac{(\tan\phi)(b^2m + J)}{\gamma}v_2\right) \tag{3.19}$$

The remainder of the controller falls out as the full-state controller did above. Again, this controller has only been designed in hopes of creating a more practical controller to be used on the vehicle, which has very limited memory storage. Prior to the application of this controller, analysis needs to be done on it's stability.

Chapter 4

Mathematical Simulation and Analysis

4.1 Overview

This chapter lays out the simulation environment created in $MatLab^{TM}$, and it's results. The simulation was designed to match the car's actual environment as closely as possible, and in an effort to keep true to this, the simulation used both sampling times as well as digitized data. The programs presented here have been modified from their original kinematic simulation in [6] to include the dynamics of the vehicle.

The following section will dissect the simulation into its primary programs. In addition to the inclusion of dynamics, other modifications have been made in order to generalize the algorithm, and will be noted at the appropriate time.

4.2 Simulation Program

4.2.1 Initialization

There are two files involved with the initialization of the simulation. One, `constants.m`, is simply all the constants involved with the vehicle such as mass, wheelbase, number of sensors, and motor and servo constants. It is this file that the user chooses the desired velocity of the vehicle. The second file, `initial.m`, creates the path for the vehicle to follow, sets the car on the path, allows the user to choose which controllers to implement, and sets what output is shown.

In order to create the path, `initial.m` calls another file, `F_path.m`, which can be adjusted to use any continuous, time-invariant function. This has been changed from the original path-creation function, which assumed the path to be either a straight line or circle. The path is defined in the original (x, y) coordinate system, with the function being of the form $y = f(x)$. For this thesis, the function $y = x^3$, Figure 4.1, has been chosen as the test path.

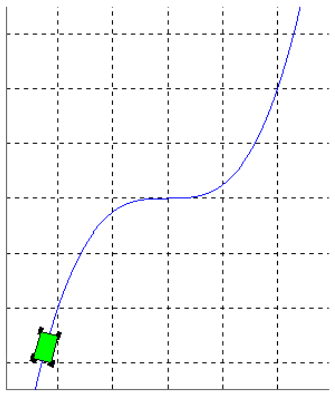

Figure 4.1: Path to traverse, with car

Once the vehicle is set on the path, the main program, `model2.m`, creates the control equations symbolically and then starts the dynamic simulation loop.

4.2.2 Error Calculation

The first part of the loop is the error calculation. The actual vehicle is outfitted with sensors located at the front and rear axles. From these sensors, the vehicle finds its lateral error from the centerline of the path. With this bit of information, it derives it's relative error in orientation, θ_p, and it's relative distance, d, from the path.

Actual Error Calculation

Since the vehicle's position, (x_0, y_0), and orientation, θ, are known, the position of the front axle is simply defined as

$$\begin{aligned} x_1 &= x_0 + l\cos\theta \\ y_1 &= y_0 + l\sin\theta \end{aligned}$$

With the center positions of the front and rear axle's known, the lateral error from the path can be found. Using figure 4.2 as a reference, we can define the slope of `LINE1` as $\tan(\theta)$.

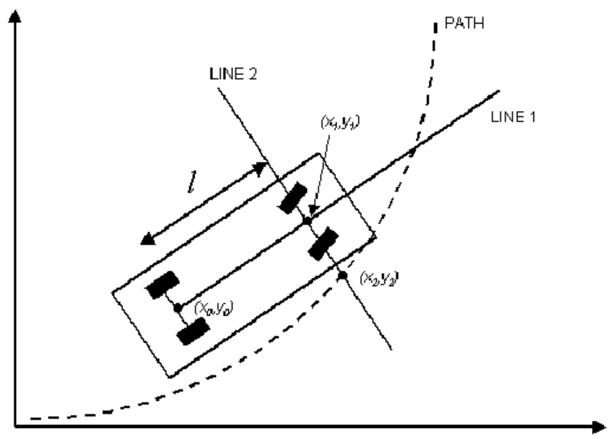

Figure 4.2: Graphical interpretation of the vehicle's error [6]

Since the sensor arrays are positioned perpendicular to `LINE1`, the slope of `LINE2` is simply $\tan\left(\theta - \frac{\pi}{2}\right)$. Now that the slope and a point on `LINE2` are known, its equation can be written as

$$y = mx + (y_1 - x_1 * m)$$

By setting the right side of the above equation equal to that of the path, the point (x_2, y_2) may be found. In the original code, this was done numerically. Since the path function has been generalized, this point is found by using *MatLab*'s symbolic library. With the two points (x_1, y_1) and (x_2, y_2), the lateral error can be found by finding the magnitude of the resultant vector.

$$e_f = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

The sign of the error is dependant upon which side of the vehicle the path falls. If the path is on the right side of the vehicle, the error is considered positive. The same method is used for the rear error, e_b, by translating `LINE2` to (x_0, y_0).

Discretized Error Conversion

The errors, e_f and e_b, are the exact distances of the axles from the path. As stated previously, it is the intent of the simulation to model, as closely as possible, the vehicle and its equipment. The car uses an array of sensors that are either on or off, depending on the position of the line. For instance, if the car was slightly to the left of the center of the path, the resultant sensor reading would be

111111100111

where the zeros show the position of the line beneath the vehicle. As a result, the errors found above must be digitized. This is done in the function `sensor.m` by the following.

```
array = ones(s,1);

for k = -0.5*s:0.5*s-1
   if (d >= k*space) & (d <= (k+1)*space)
      array(s/2-k) = 0;
   end
end
```

In this function, `s` is number of sensors, `d` is the error, `e_f` or `e_b` from above, and `space` is the sensor spacing. From here, the error is converted back into a distance by

```
error = 0;
num = 0;
val = (s+1)/2;

for k = 1:s
   if array(k) == 0
       num = num+1;
       error = error+(val-k)*space;
   end
end

if num ~= 0
   error = error/num;
else
   error = B_w/2*sign(p);
end
```

`B_w` above refers to the width of the sensor array and `p` refers to the previous error. The final if statement in the above code has been placed in the event the line does not appear underneath the vehicle. In that event, the vehicle assumes the error to be the maximum value and that the line is located on the same side of the vehicle as it was last time.

Relative Heading

With the lateral errors of both the front and rear of the vehicle known, the heading angle with respect to the path, θ_p, may be found. For this, the algorithm assumes the path beneath the vehicle to be straight, and the resultant heading is

$$\theta_p = \tan^{-1}\left(\frac{e_f - e_b}{l}\right)$$

In the simulation, the descritized errors are used rather than the actual errors.

4.2.3 Real-time Modeling and Animation

In order to implement the controllers designed in the prior chapter, a few variables must be known. These variables are d, the lateral deviation of the rear axle, θ_p, the error in the heading of the vehicle, $c(s)$, the curvature of the road, and v_u, the velocity of the vehicle. The acquisition of the first two variables was just discussed. There are many ways in which the curvature can be measured, and are the focus of the thesis written by Patricia Mellodge. With this in mind, the curvature is assumed to be known by the vehicle. The vehicle also has an encoder located on the rear axle of the car, and, therefore, the velocity is known.

With these four variables, all of the states can be evaluated. Once the states are known, the control equations can be evaluated, whose outputs are the steering and velocity inputs. These inputs are then passed to a *Simulink* model of the dynamics of the system, Figure 4.3.

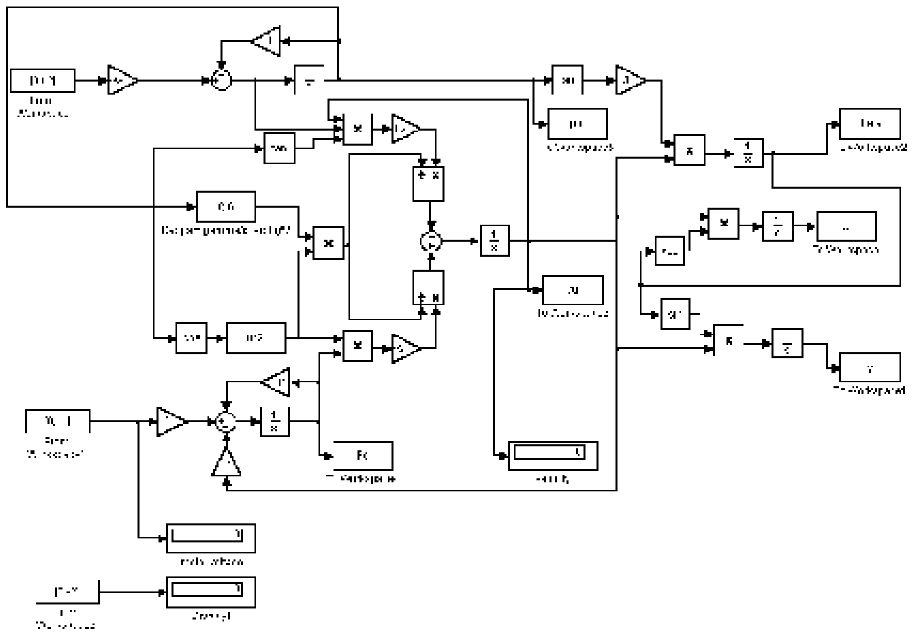

Figure 4.3: Simulink chart of the car's dynamic model

The model operates for a time T, which is the sampling period of the vehicle's sensors. The initial conditions given to the model are the current states of the vehicle, $(x, y, \theta, \phi, v_u)$. The steering and velocity inputs are constant for the full period of the simulation, resembling the actual car. At the end of the period, the final states of the model are passed back to `model2.m` to allow the controllers to be updated, and the loop starts over at the error calculation.

Upon each iteration of the *Simulink* model, a frame is captured using *MatLab*'s animation toolbox. This part of the code was designed in order to monitor the behavior of the vehicle as it traversed it's path. The animation functions were designed by Patricia Mellodge, and were the only functions that were used as-is in this research.

4.3 Simulation Results

4.3.1 Results of the Velocity Controllers

Since the speed controller was more involved in its derivation, it will be discussed first. Prior to this thesis, there has not been a focus on the speed controller of the vehicle. As a result, there was nothing to compare these results to. Since this is the case, certain requirements have been set. Specifically, the maximum overshoot of the velocity should be no more than 5% of the steady-state speed. Further, the controller should react as quickly as possible to changes. Finally, the test car has a restricted power source, which is 5 volts. As a result, a third requirement was made to ensure that the control algorithm would not call for more power than was available. After some testing, an optimal set of gains, based on the above requirements, were found. The first velocity controller reviewed is the full-state controller. The response of this controller is shown in Figures 4.4 and 4.5. As figure 4.4 shows, the

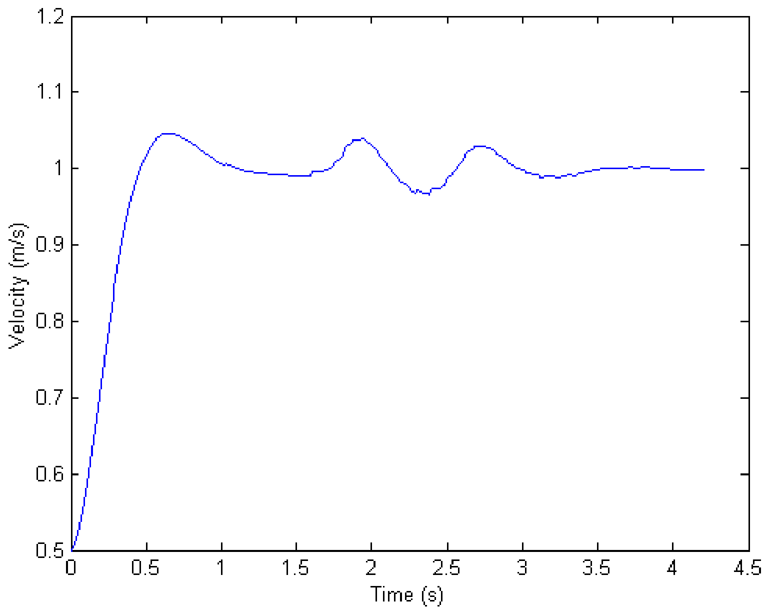

Figure 4.4: Velocity response of the car

settling time for the velocity is approximately half a second. Furthermore, the maximum

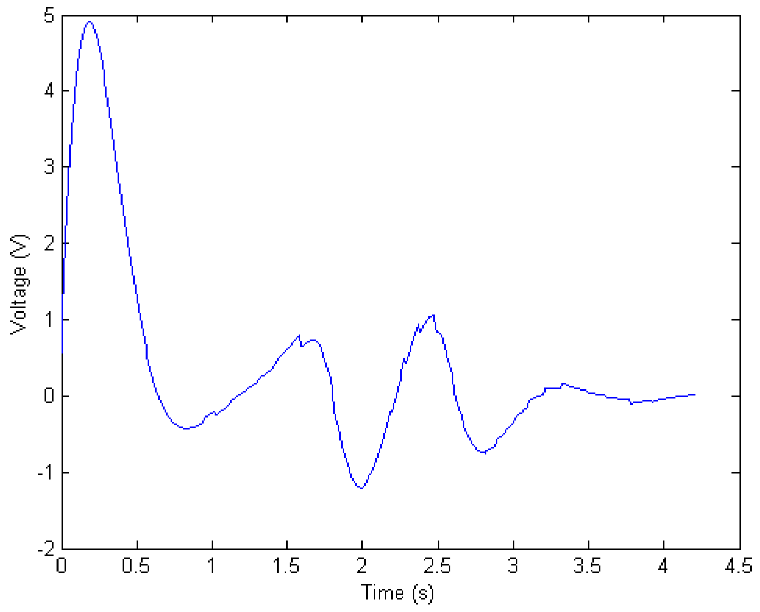

Figure 4.5: Required control effort

overshoot is 4.6%. As figure 4.5 shows, the voltage to the motor does not reach 5 volts, which means that the controller is effectively controlling the vehicle without overloading the motor or the battery. The original intent was to show the effect of the truncated controller on a separate graph, but after running the simulation, a very interesting outcome occurred. As a result, figures 4.6 and 4.7 show both controllers plotted together. The full-state controller is in blue while the truncated controller is in red. With such a minimal change in response, the control signal was next in line to be inspected. Figure 4.8 displays the difference in the full-state $\dot{\eta}$ and the truncated version. At first, this caused more confusion than good, until the two of these were plotted against v_1, the other half of the control signal, equation 3.14. Figure 4.9 explains the quite obvious reason for such small change in the response.

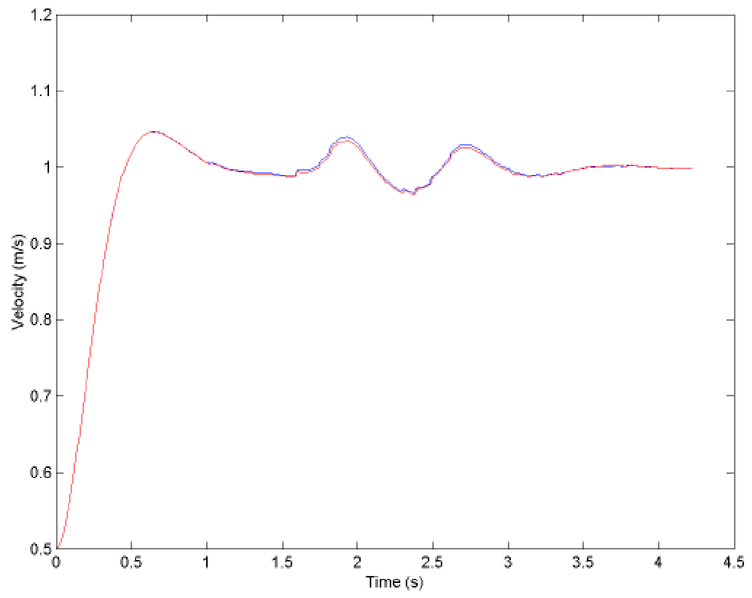

Figure 4.6: Comparison of velocity response of the car

4.3.2 Results of the Lateral Controller

While the velocity controller was the majority of the derivation presented here, the effects of the servo dynamics were quite surprising. In order to see these effects, the simulation was run twice. Both simulations use the velocity controller designed here, but while one lateral controller takes the servo dynamics into account, the other uses the current kinematic controller being used on the vehicle. All the plots to follow have the dynamic responses in blue, with the kinematic response in red. The difference in the resultant velocity curves, figure 4.10, alone is astounding.

While difference in the velocity response is impressive, the main goal of the lateral controller is to keep the vehicle on the path. With this in mind, the most important measurements of success are the effect on the error in the orientation of the vehicle and the lateral error of the wheels with respect to the path. Figure 4.11 shows the resultant heading errors of the two controllers.

At first glance, the errors of the front and rear axles, figure 4.12 seem to not be any better. But, when the sum of the magnitude of the errors, figure 4.13, are compared, it is evident that the dynamic controller gives better results.

These plots show that, while the path is mostly straight, the kinematic method is sufficient to control the vehicle, but when the curve gets tighter, it does not respond as well as the

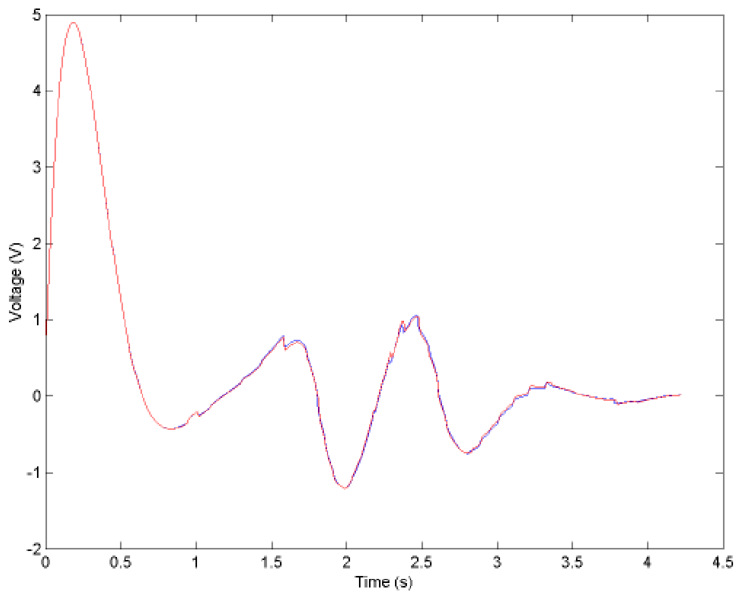

Figure 4.7: Comparison of control effort

dynamic controller. Possibly the most interesting observation of the result is that, while the majority of the plots show the kinematic model to produce more error, the error experienced at the rear of the vehicle is actually less, nearly zero. On the other hand, using only the kinematic model causes the velocity to nearly double in order to get the vehicle back on the track, which in turn, means that the control effort is much higher. Furthermore, the error experienced at the front of the vehicle is more than 50% greater when the dynamics of the servo are ignored.

All the files used to create this simulation, and the animation, are located in Appendix B.

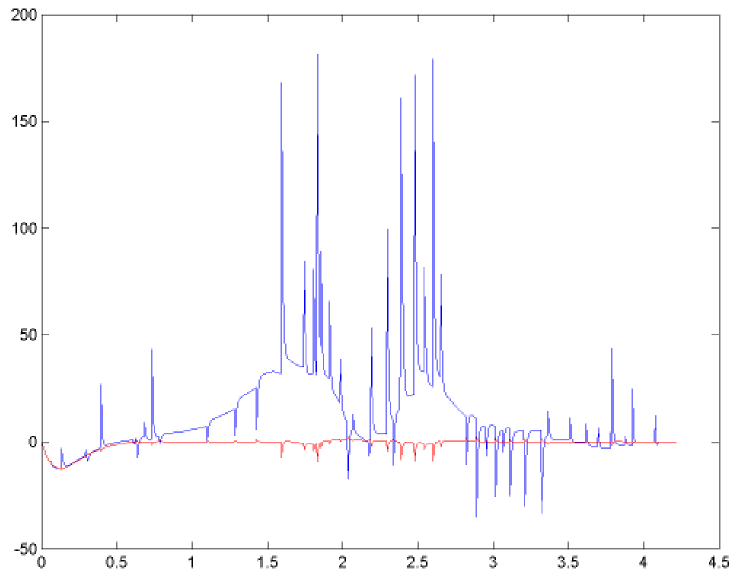

Figure 4.8: Full-state $\dot{\eta}$ vs Truncated $\dot{\eta}$

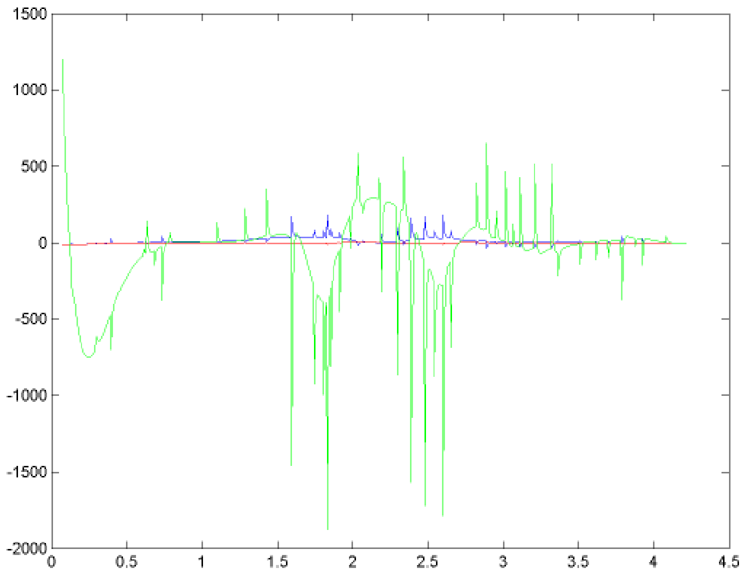

Figure 4.9: Comparison of v_1 versus $\dot{\eta}$

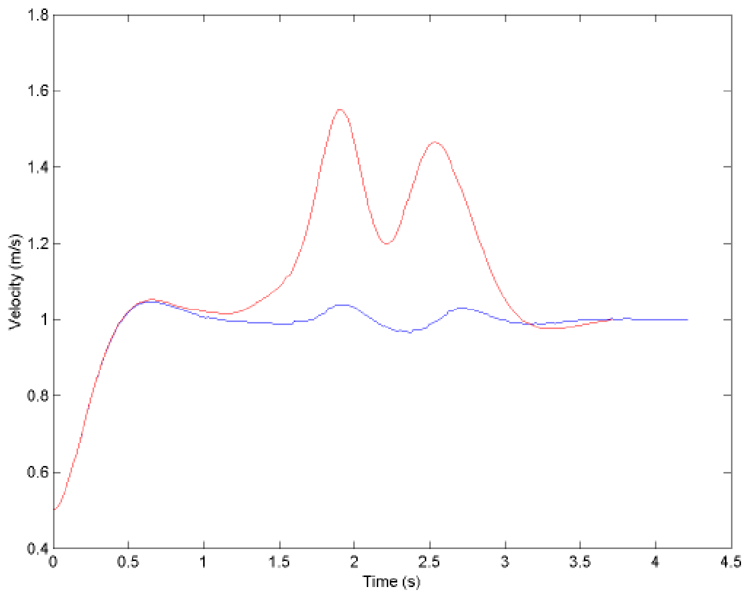

Figure 4.10: Comparison of velocities resulting for inclusion of servo dynamics

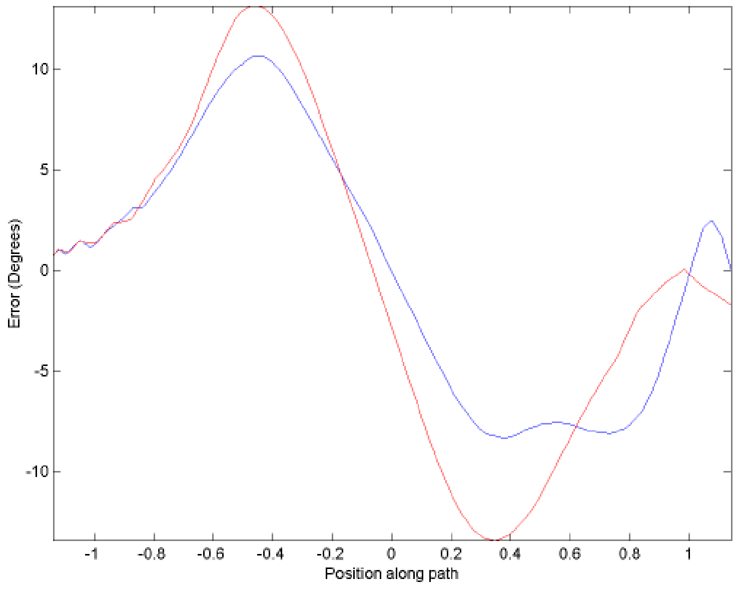

Figure 4.11: Comparison of orientation errors

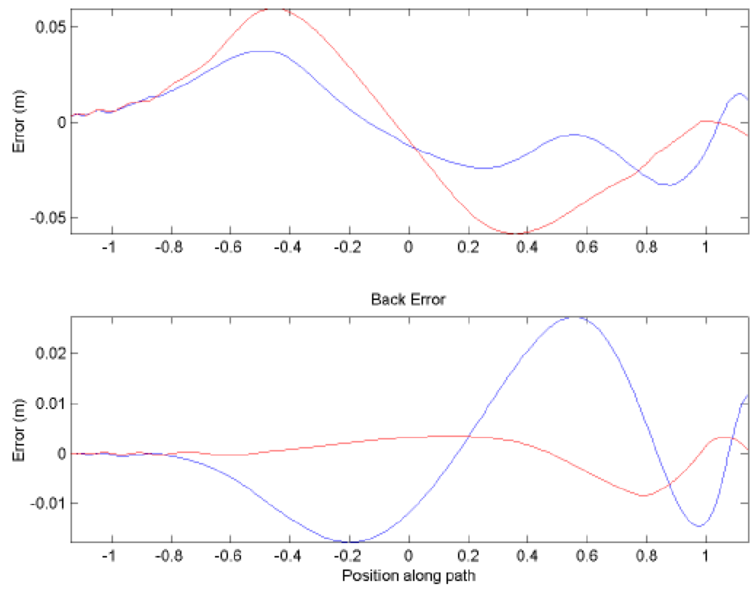

Figure 4.12: Comparison of front and back errors

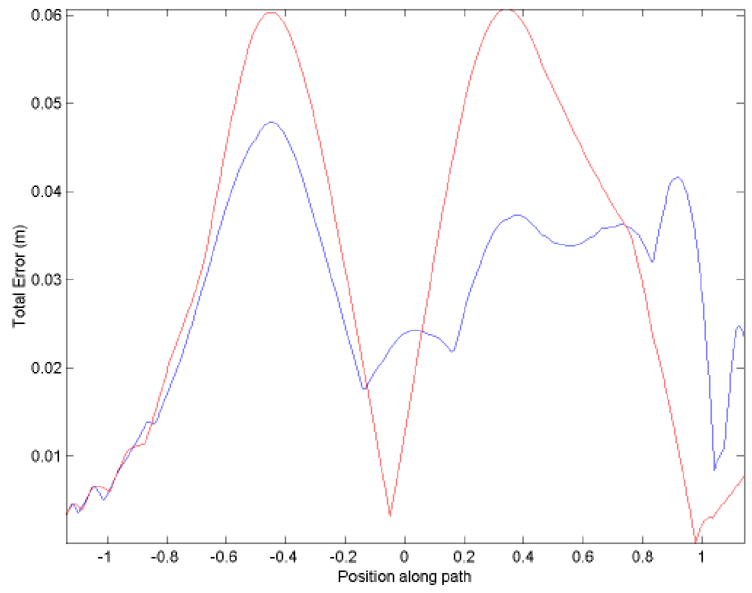

Figure 4.13: Comparison of total error magnitude

4.4 Mathematical Analysis

When looking through the results, questions might arise with respect to significance of the parameters. Specifically, one might ask, ”what happens to the impulses seen in the intermediate steps of the velocity controller?” This will be explained first. The remaining portion of this section will focus on two parameters in particular: the time constant associated with the steering servo and the sampling frequency of the road with respect to the vehicle. The effect of the steering time constant will be explored via the comparison between the error in orientation and the vehicle's speed response of the kinematic and dynamic steering controllers. When the sampling frequency is considered, the full dynamic controller will be used, and its response will be compared at different sampling rates.

4.4.1 Evaluation of Speed Controller

When one compares Figures 4.9 and 4.4, or 4.5, there seems to be a discrepancy or two. Figure 4.9 shows that (1) the controller has quite a few large spikes throughout simulation and (2) the magnitude of the controller is quite large, on the order of 10^3. In comparison, the output shows very little sharp changes and is on the order of 10^0. Both of these phenomena can be explained by referring back to Equation 3.10.

$$u_1 = \frac{R_w N_m L_a}{N_w K_m}\left(w_1 + \frac{(R_a b_m + K_m K_b)\,N_w^2}{N_m^2 R_w^2} + \frac{R_a}{L_a} F_D\right)$$

The first inconsistency can be explained by examining the gain associated with the feedback of the driving force, $\frac{R_a}{L_a}$. The values of the resistance and impedance were experimentally found to be 1.9 and $1.064e^{-4}$, respectively. The resultant gain on the driving force is on the order of 10^4. This leads to the driving force washing out the majority of the spiking visible in the controller. The solution to the second misunderstanding is can be solved in the same way. Specifically, the gain of the entire function, $\frac{R_w N_m L_a}{N_w K_m}$, has a magnitude on the order of

4.4.2 Effect of the Steering Time Constant

The effects of the servo dynamics have been shown to cause a significant change in the overall response of the vehicle. The next step is to explore when these dynamics have an effect and when they can be ignored. As a result, the simulation was run with the time constant of the servo, τ_s, ranging from 0.025, half of the actual value, to 0.325 in steps of 0.050. As expected, the dynamic controller was virtually unaffected by the change in the servo response speed, Figures 4.14 and 4.15.

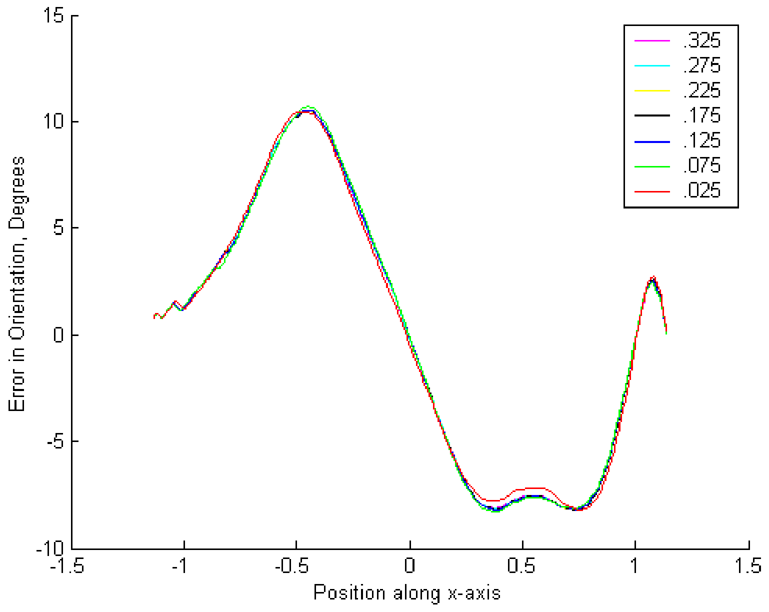

Figure 4.14: Dynamic Error in Orientation w.r.t. Changing Time Constants

The kinematic controller, on the other hand, showed some interesting results. The most notable is that, as the time constant for the servo increased, the amount of error decreased. Mathematically, this makes sense because as the time constant increases, it decreases the effect of the current state on the response. These results are shown in Figures 4.16 and 4.17.

Furthermore, these plots show that, as the time constant increase, the kinematic response converges towards that of the dynamic response. This is made more clear when the kinematic and dynamic responses are plotted against each other for the time constants of $\tau_s = 0.025$ and $\tau_s = 0.325$. These plots can be seen in Figures 4.18, 4.19, 4.20, 4.21.

It is worth mentioning that a time constant of 0.325 correlates to a settling time of roughly 1 second. The settling time for the servo used on the vehicle is on the order of 0.16 seconds. With this in mind, and the fact that the kinematic controller has yet to give a response equivalent to that of the dynamic controller, leads to the conclusion that including the dynamics of the servo is vital to a high quality controller and is not dismissible for any realistic values of τ_s.

4.4.3 Effect of Sampling Rate

Finally, the effect of the sampling rate on the response of the vehicle was given more scrutiny. As Figures 4.22 and 4.23 show, the sampling frequency makes a negligible difference in terms of keeping the vehicle centered on the road, with one exception. In keeping with the Nyquist

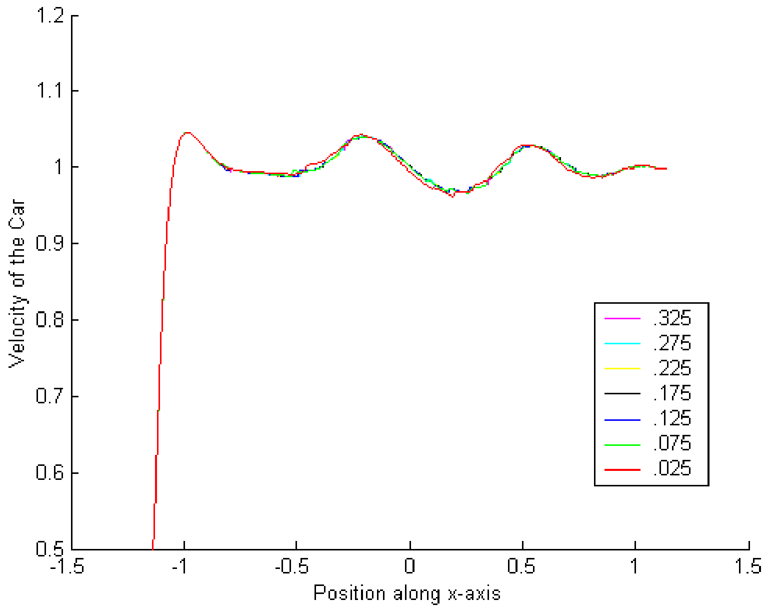

Figure 4.15: Dynamic Speed Response w.r.t. Changing Time Constants

theorem, if the sampling period is increased from 0.025 seconds, the system becomes unstable due the control frequency of the servo, whose period is approximately 0.054 seconds.

Unlike the lateral controller, the speed control experiences more change as the sampling rate changes. As Figure 4.24 shows, the sample period of 0.025 seconds is already sending the system towards an unstable solution. More importantly, however, is the response that is being converged upon as the sampling rate increases. At a frequency of 1kHz (T = .001), the velocity response reaches steady state, does not overshoot, and is virtually flat for the remainder of the simulation.

Now, the spiking found in the intermediate steps of the speed controller will be readdressed. While the reason for its lack of appearance in the final output has been explained, the reasons for their existence have not. These intermediate steps are derived from systems that are not only nonlinear; they are also not continuous. η, 3.12, its derivative, and all the resultant steps in the controller have a large number of discontinuities in them. These discontinuities include both magnitude and sign of variables. The good news is that as the sampling rate increases, the magnitude of the spikes decrease, Figures 4.25 and 4.26.

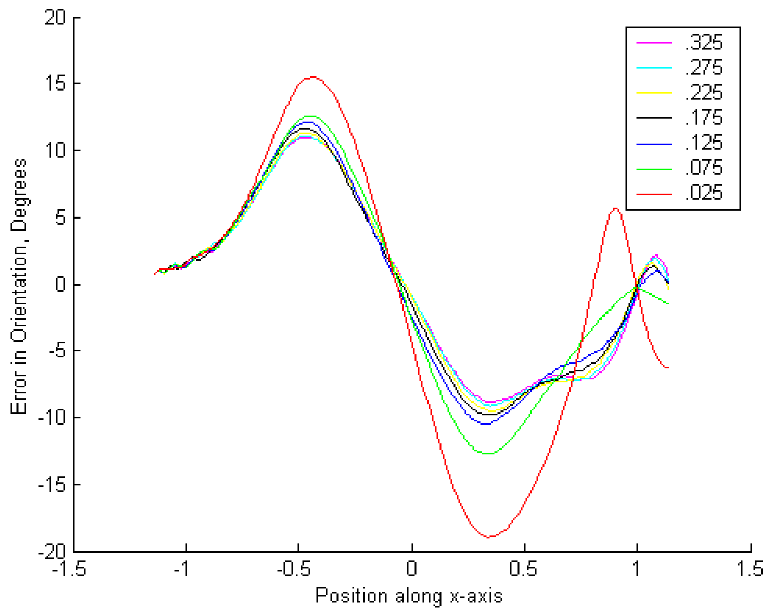

Figure 4.16: Kinematic Error in Orientation w.r.t. Changing Time Constants

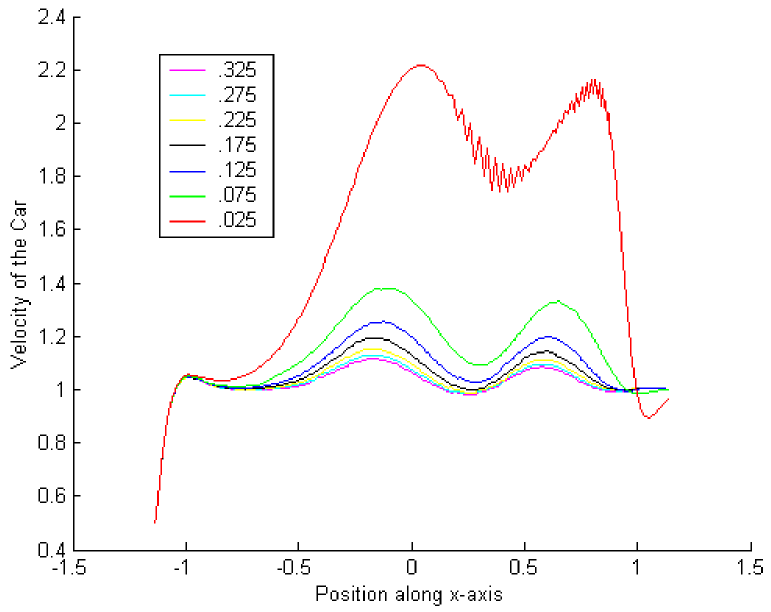

Figure 4.17: Kinematic Speed Response w.r.t. Changing Time Constants

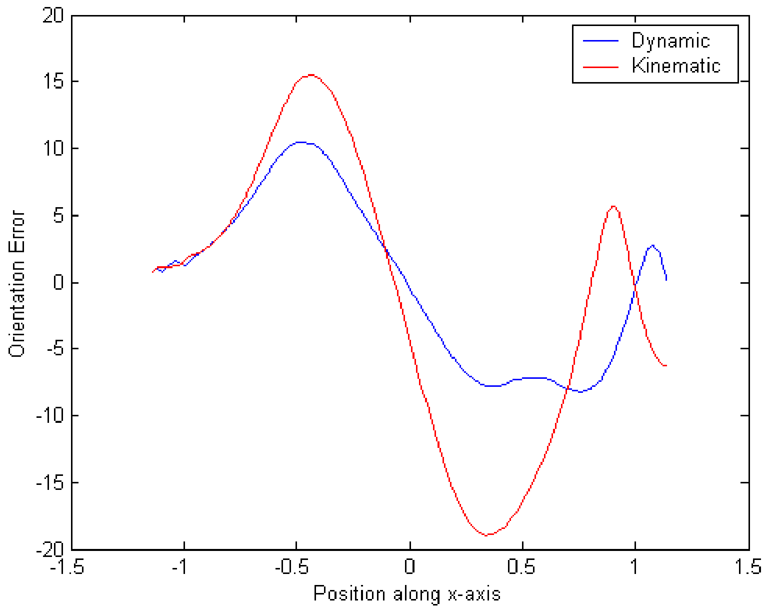

Figure 4.18: Kinematic vs Dynamic Response in Orientation for $\tau_s = 0.025$

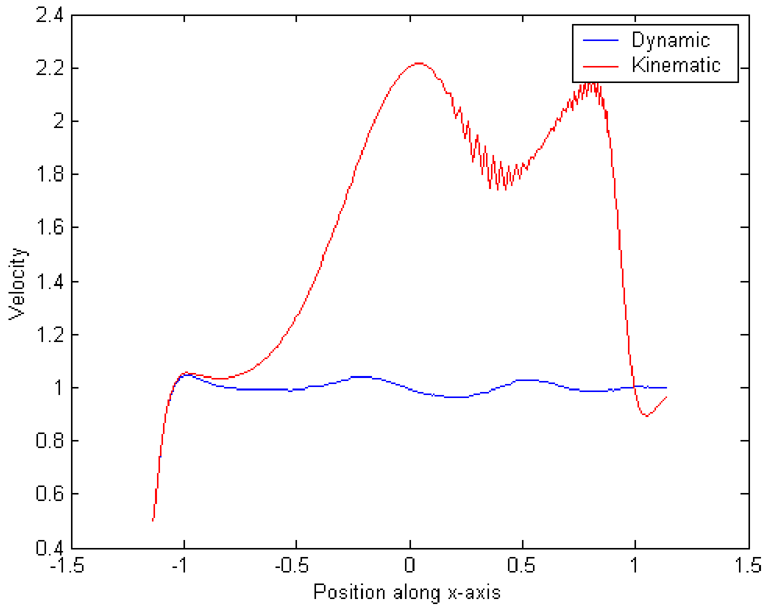

Figure 4.19: Kinematic vs Dynamic Response in Speed for $\tau_s = 0.025$

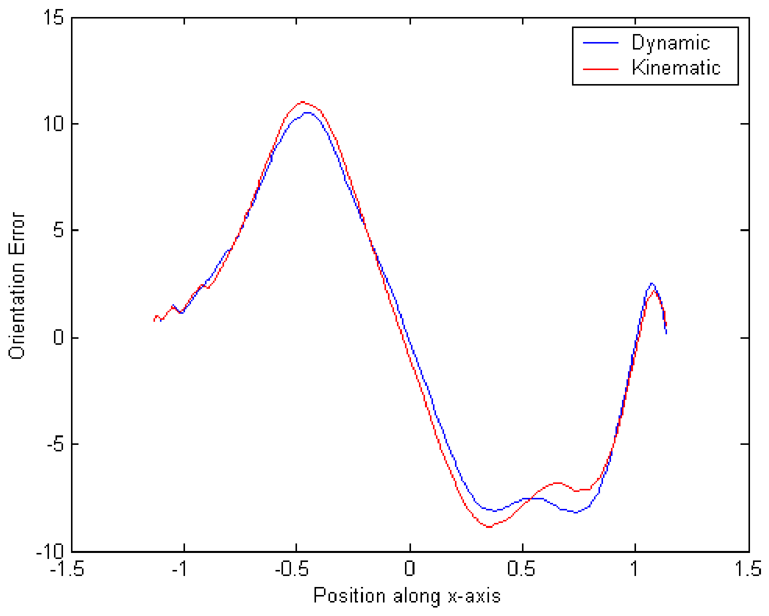

Figure 4.20: Kinematic vs Dynamic Response in Orientation for $\tau_s = 0.325$

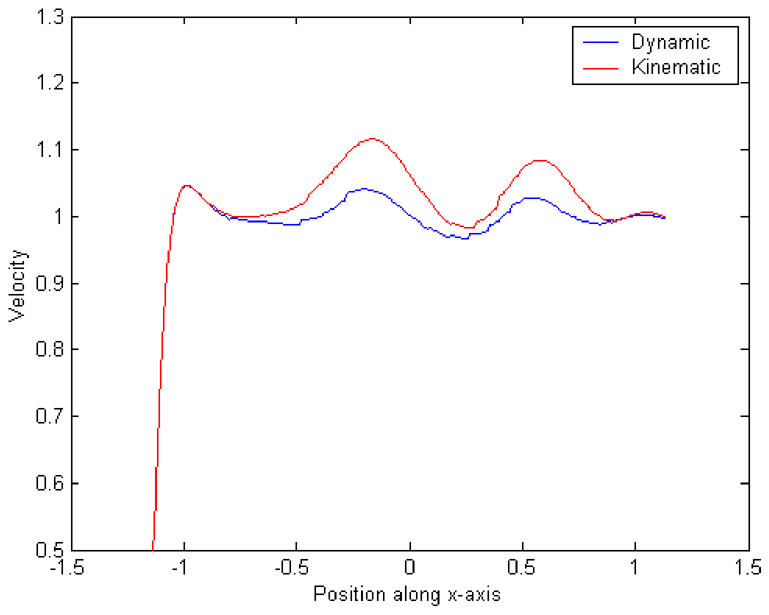

Figure 4.21: Kinematic vs Dynamic Response in Speed for $\tau_s = 0.325$

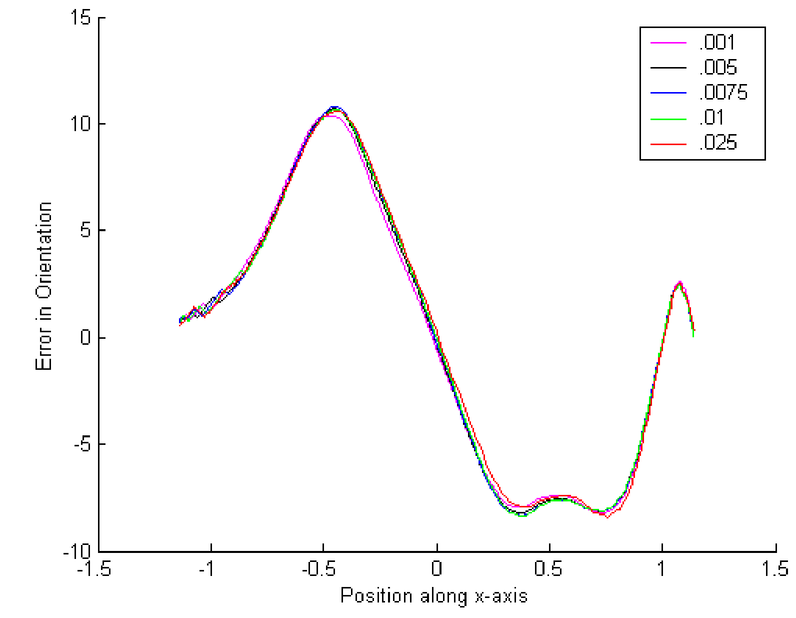

Figure 4.22: Error in Orientation w.r.t. Sampling Period

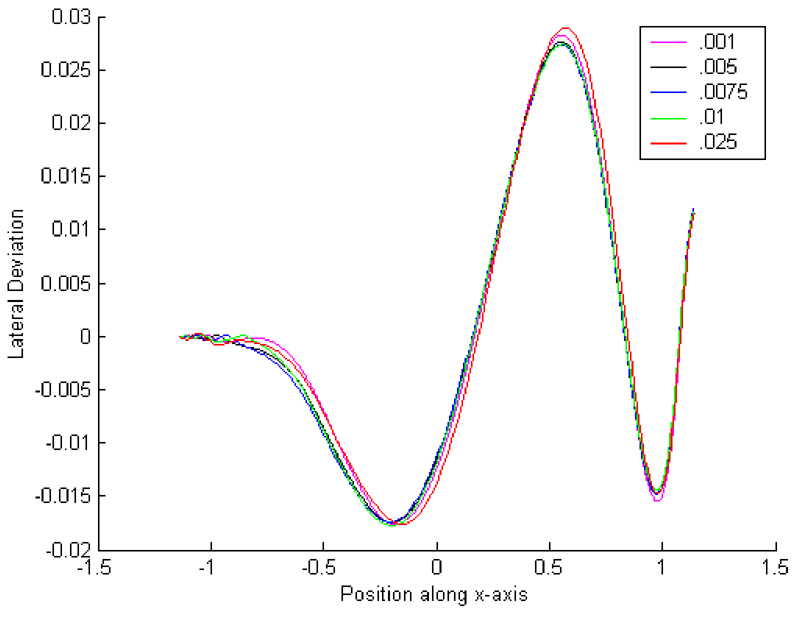

Figure 4.23: Lateral Deviation w.r.t. Sampling Period

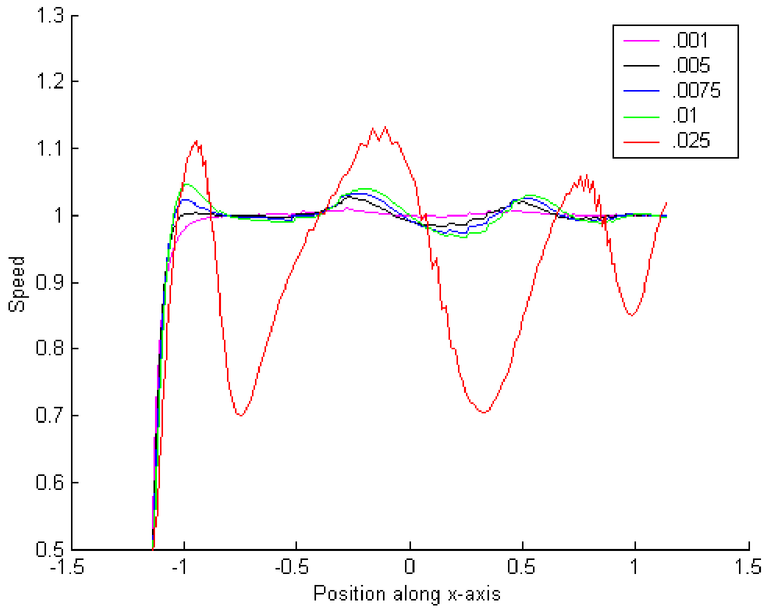

Figure 4.24: Speed Response w.r.t. Sampling Period

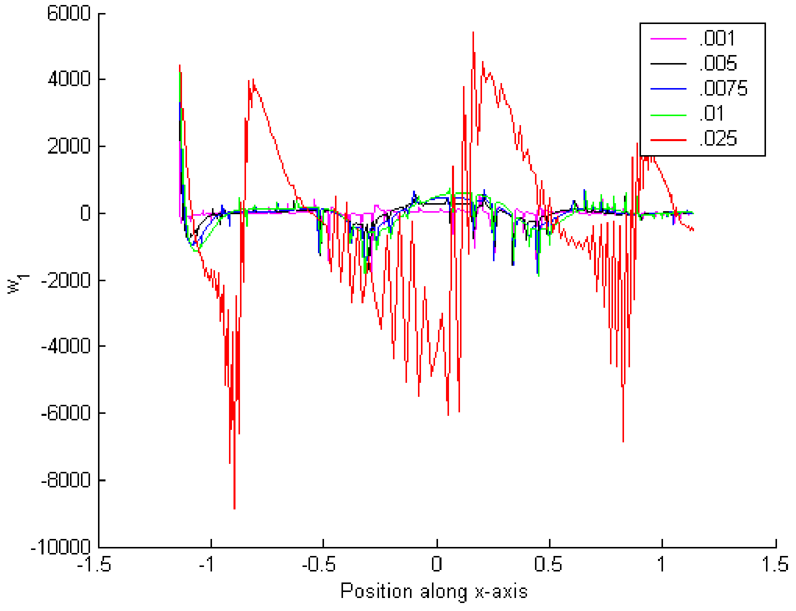

Figure 4.25: Speed Response w.r.t. Sampling Period

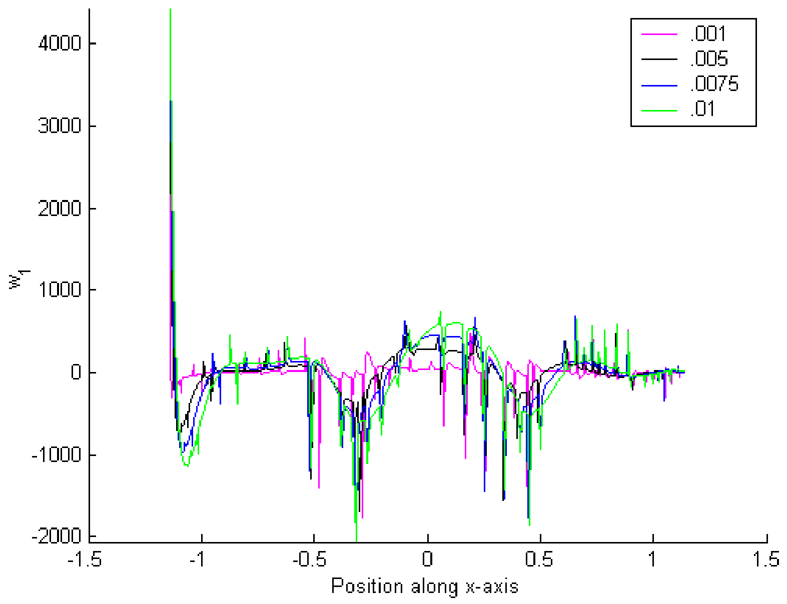

Figure 4.26: Speed Response w.r.t. Sampling Period

Chapter 5

Hardware Platform

The prototype vehicle is composed of wide range of components from sensor arrays to microprocessors to basic remote-control car parts. In order to keep in the ”low-cost” scope of the project, the majority of the hardware is off-the-shelf components. The high-level block diagram for the vehicle is shown if Figure 5.1.

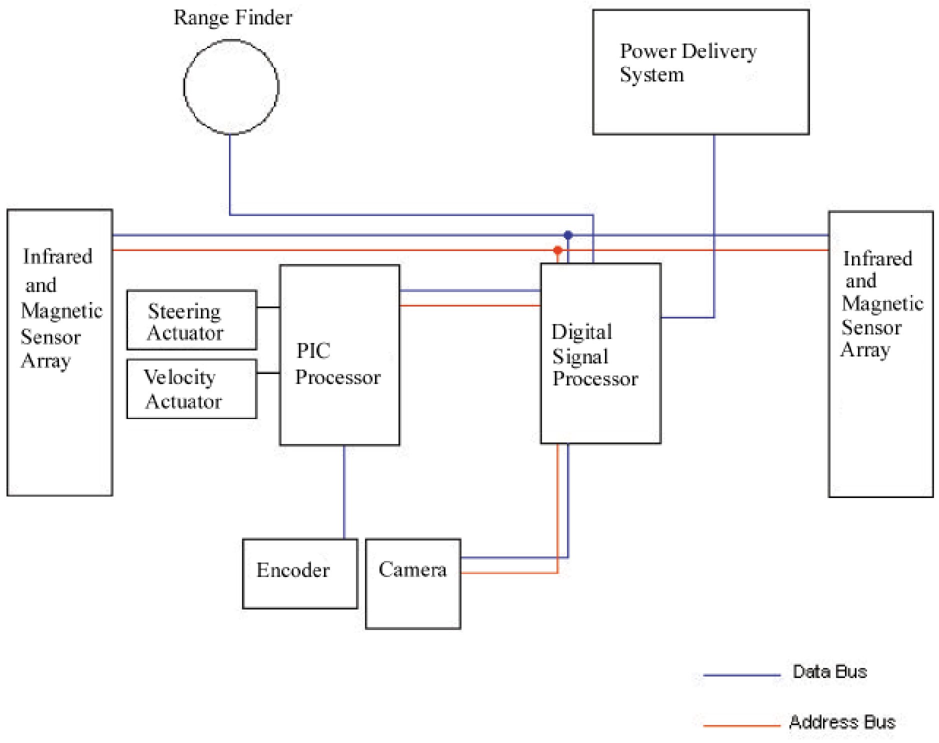

Figure 5.1: High-level block diagram of the hardware platform

The primary components of the vehicle are the scale vehicle platform, the Digital Signal Processor (DSP), the PIC processor, an the sensory equipment consisting of infrared, magnetic, range-finding, and vision. The easiest way to describe the hardware platform is with respect to a driver behind the wheel his car. The DSP acts as the ”brains” of the system. It receives information from the host of sensor arrays, or ”eyes”, and decides what must be done. At this point, it sends signals to the PIC, which acts as the ”arms and feet” of the system, which controls the motor and steering of the vehicle.

5.1 Scale Vehicle Platform and Actuators

The foundation of each vehicle is built from basic remote-control (RC) components that can be purchased from any hobby shop. The chassis is a 1/10 scale RC car kit. Included in this kit is the carbon-fiber frame, all the drive and steering hardware, and a steering servo. The motor and battery are not included in the kit, but since they are off-the-shelf products, they can be purchased at the hobby shop as well. Table 5.1 shows the specific components being used on the current prototype.

Table 5.1: RC components currently in use on the FLASH car.

Component	**Manufacturer**
Legends 1/10 scale model car kit	Bolink
Steering servo	Futaba
Paradox rebuildable electric motor	Trinity
7.2V NiMH battery	Radio Shack

Since standard RC motors are made for speeds in excess of 300MPH (scale) and the focus of this project is to operate at speeds to not exceed 65MPH, the motor has been re-wound. The current motor is wound with 100 turns of 30 AWG wire, which limits the maximum speed of the vehicle, as well as reducing the power used by the motor. In earlier versions, a *Novak* speed controller was used, but this has been replaced by an h-bridge for more precise control over the vehicle's velocity. Not only does the h-bridge give more precision in the velocity, it also gives a full range of reverse, which was not available with the *Novak*. To operate, the h-bridge requires two signals. The first is a direction, which is either high (5V) for forward, or low (0V) for reverse. The second is a pulse-width modulated (PWM) signal, operating at approximately 1.4kHz. These two signals are sent from the PIC processor, and updated at approximately 5kHz.

In order to keep the speed of the vehicle constant, a *U.S. Digital* HEDS-90041 optical encoder has been placed near the rear axle. The encoder disk of 512 counts per revolution in mounted directly to the rear axle. As the axle turns, the encoder sends a pulse to the PIC. The PIC then counts the pulses over a set period of 14ms, and sends the count to the DSP. The DSP then uses a simple conversion from encoder ticks to meters per second to determine that actual speed of the vehicle. The DSP then adjusts the speed of the vehicle accordingly. The speed controller will be discussed in more detail in the controller and software platform chapters.

The servo is controlled by a servo control pulse (SCP) whose period ranges from 10ms to 15ms. A standard servo can produce angular rotations in excess of 135 degrees. The position of the servo is controlled by the high period of the SCP, which ranges from 0.75ms to 2.25ms, with a nominal period of 1.5ms for the "center" of the servo's range. The functional range of the steering servo on the vehicle is limited to 90 degrees due to physical restraints. As a result, the high period sent to the servo is limited to a range of 1ms (full left) to 2ms (full right), with the center still being 1.5ms. This signal is also sent from the PIC processor.

5.2 Digital Signal Processor

The majority of the processing, as well as the main computing power behind the car, is a digital signal processor made by *Texas Instruments.* A few of the specifications for the latest DSP are

Table 5.2: Specifications for the C6711 DSP.

Full Device Name	TMS320C6711C
Manufacturer	Texas Instruments
Processor	32-bit Floating Point
Operating Voltage	3.3V
Operating Frequency	150 Mhz
Cycle time	5 ns
Serial Ports	2
DMA channels	16
Timers	2 - 32 bit

One of the reasons the project decided to go with *TI* was because you can get a starter kit (DSK) along with the chip, making interface much easier. The DSK has a parallel port on-board, and coupled with Code Composer, an interface for the DSK, *TI* allows the user to interface the DSP directly. This allows for easy debugging of the program, as well as easy loading of the program into the on-board ROM.

As stated before, the DSP can be viewed as the "brains" in the operation. Keeping this in mind, there are 3 major functions of the DSP : data collection, data processing, and control signaling.

5.2.1 Data Collection

There are two directions the DSP will receive data from its peripherals. The major path is through the parallel data bus, and the other, less used, is the DSP's serial ports.

Parallel Data Bus

The DSP receives information from the infrared line-detectors, magnetic sensors, and the PIC from its data bus. While the data is, obviously, passed on the data bus, the individual peripherals must be called. In order to do this, a small portion of the address bus is brought into the picture. Figure 5.2 gives example to how the DSP is connected to its devices.

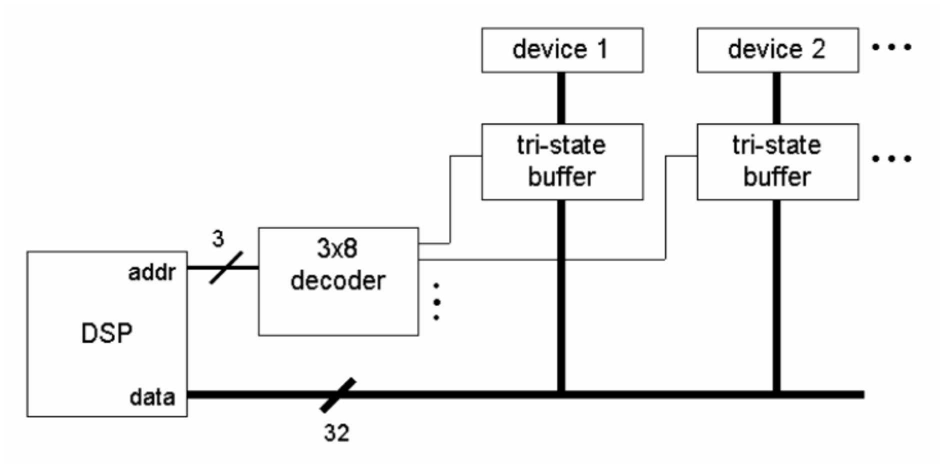

Figure 5.2: Connection of DSP to its peripherals [6].

As the figure illustrates, each device is buffered so that signals from one set of devices does not interfere with those from another set. The 3 LSBs of the address bus are used to cycle through the devices by enabling and disabling the buffers. In this way, the devices can update their information as often as they like, and the only lag in receiving the information is due to the access time of the 3-to-8 decoder. Using the decoder allows a total of 8 peripherals to be accessed by the DSP. Currently, there are 5 devices being accessed by the DSP, which leaves room for future additions.

Serial Port

The DSK has been designed and configured to read in analog signals through an on-board A/D/A converter via its serial port. There are currently two analog signals being received by the DSP. These signals are the output from the range-finder used for adaptive cruise and the battery voltage. Soon to be added to this list will be the current to the motor. The A/D converter has 6 channels available, so receiving data from the separate devices is as easy as that of the data bus.

5.2.2 Data Processing

Once the data is received from the various sensor arrays, it must be processed. What this means depends on which sensor's data is being processed. As a result, this section has been divided into the following sections: line detection, curvature estimation, headway detection, velocity detection, health monitoring, and control.

Line Detection

There are two major sensor arrays involved in line detection: infrared and magnetic. Both arrays send the same form of data to the DSP, which is why they will be covered as a single unit. These sensor arrays consist of bumpers on the front and back of the vehicle. Each bumper is outfitted with a set number of sensors, N, which ultimately send N bits of data to the DSP. Since there is a front and rear bumper, the total number of bits sent to the DSP is $2N$. Normally, these bits are 5V, but if the line, be it visual or magnetic, is below the sensor, the bit is pulled to 0V. Once the DSP reads in this information, it separates the front bumper from the rear and arranges the bits in order such that the left-most sensor corresponds to the most significant bit and the right-most, the least. Once in this order, it is treated in much the same way as it was presented in the simulation section of this thesis. The total deviation from the center of the path can be represented as

$$d = \frac{\sum_{i=0}^{N-1}(k-i)\Delta x}{N_{on}}$$

where k is $N/2$, i is the bit's position, Δx is the sensor spacing, and N_{on} is the number of sensors that are turned on.

Curvature Estimation

There are two ways in which the curvature of the road can be estimated. One of these methods is to use the output from the line detection and approximate the curvature, based on one of the techniques described in [6]. The other is to use image processing. While the image-processing algorithm has been designed, it has not been implemented at the time of this writing. The reason for this is that the previous DSP was not capable of it, and so it has been on hold. Hopefully, it will be online by 2003.

Headway Detection

In previous versions of the car, an ultrasound sensor was used in order to find the headway. Due to its power consumption and instability, it has been replaced by an infrared range-finder. Both sensors output a similar signal, which is analog. This analog signal is read into the A/D converter, and then sent to the DSP via the serial port. The value is then converted into a distance using an exponential function, which well represents the output-distance curves given in the technical information of the sensor.

Velocity Detection

Velocity detection is handled slightly differently than the other sensors. The actual detection is done using an encoder located on the back axle, whose pulses are sent to the PIC. From there, the PIC sends the encoder count to the DSP via the data bus. Once received from the PIC, the DSP converts the number of ticks into a measure of velocity in meters per second.

Health monitoring

The most important piece of information involved with health monitoring is the battery voltage. Since the car will be autonomous when completed, it is imperative that the car knows when to recharge. The voltage is sent through a buffer, to avoid overloading other circuitry, and then into the A/D converter. In the near future, the vehicle will also start keeping track of its average deviation and orientation error. This information has already been extracted by the line detection and requires no excess input. In the event that these values get too far from zero, the vehicle will notify the rest of the system that it needs maintenance.

Control

Once all the data has been processed into useful information, the control algorithms are employed. Depending on the setup, one or all the technologies can be used. The control algorithm is a conglomeration of works including curvature estimation from [6], automatic cruise control from [7], as well as others unknown to this author.

5.2.3 Control Signaling

After the DSP has run through its control algorithms, it sends commands to the PIC in the form of 8-bit words. There are 4 distinct command words it sends to the PIC, using the 2 MSBs as addressing. The format is as follows.

Table 5.3: Format of the command words to the PIC.

Bit	**Usage**
0..5	Portions of the velocity or steering value
6	0 is steering, 1 is velocity
7	0 is low nibble, 1 is high nibble

The PIC is using a built-in 10-bit PWM generator, which allows us highly discrete control over the velocity of the vehicle. In the previous generation, the car was limited to 32 discrete velocity steps. This number is now 2048, but way of using the 10-bit PWM register as well as a direction bit. This PWM signal is updated at a rate of 2kHz, which is a big step up from 70Hz, again from the previous generation. For steering, the controller was limited to 25 discrete steps until recently. Since that time, the resolution has been increased to 156, with a value of 78 being center. Both the velocity and steering commands are split into two pieces and masked to match the format presented above. Those commands are then sent to the PIC. The values are not updated until an entire command word has been read.

5.3 Sensor Arrays

As stated above, there are quite a few sensor arrays. In this section, a closer look will be taken of each one.

5.3.1 Infrared and Magnetic Sensors

In application, both the infrared and magnetic sensors are treated identically. The reason for this is that they operate in tandem with each other to provide information as to the location of the line on the road. Both sensors are positioned so they ”look” down at the road’s surface. The mediums used to find this line, however, are quite different.

Infrared Sensors

The first thing to note is that these sensors differ from the range-finding sensor which will be discussed later in this chapter. These sensors are composed of an emitter-detector pair and are specifically designed as reflection sensors. In this case, the more light that is reflected back to the detector from the emitter, the more open the detector transistor is, see figure 5.3. The output, V_{out} is the data sent to the DSP. Table 5.4 displays all the technical information about this specific sensor.

Table 5.4: Technical information for the infrared sensor.

Device name	QRD1114
Manufacturer	Fairchild Semiconductor
Emitter forward current	50 mA max
Emitter forward voltage	1.7 V
Peak emission wavelength	940 nm
Sensor dark current	100 nA
Package size	0.173" x 0.183" x 0.240"

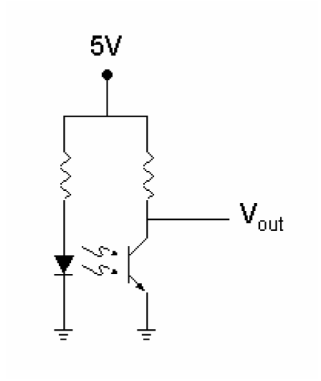

Figure 5.3: Circuit application for the IR sensor [6].

These sensors, while extremely reliable under controlled circumstances, are very susceptible to sunlight and debris on the road surface. More important to note is that, without major changes in the infrastructure of the roadways today, these sensors will not be viable because it requires the line to be underneath the car rather than beside it.

Magnetic Sensors

Much like the infrared sensors above, the magnetic sensors work by finding the position of the line underneath the vehicle. The difference is that the sensor detects invisible magnetic fields rather than light. The exact sensor used is the Hall Effects sensor, whose specifications can be viewed in table 5.5.

Table 5.5: Technical Information for the magnetic sensor.

Device name	HAL506UA-E
Manufacturer	Micronas
Operating voltage	3.8 V - 24 V
Supply current	3 mA
Switching type	unipolar (South)
B_{on}	5.5 mT
B_{off}	3.5 mT
Package size	4.06m x 1.5mm x 3.05mm

Since these sensors use magnetic fields rather than light, sunlight or other debris does not hinder their performance. While at this time, most roadways do not have magnets embedded in the road, this medium would require less change to the highway infrastructure, and will probably be one of the most viable in the future.

5.3.2 Vision

Unfortunately, the exact camera that will be used on the car is still unknown. The main use for the camera is to find the curvature in the road. Its secondary use it to assist the infrared and magnetic sensors in centering the vehicle on the path. Since the final camera has not been decided, it obviously has not been incorporated into the vehicle. The method of data acquisition, on the other hand, is known. The image will be sent to the DSP via one of the direct memory access (DMA) channels available on the DSP. From there, the image will be processed using edge-detection techniques to find the path. Finally, the converted image will be used to extrapolate the curvature and position of the path in front of the car.

5.3.3 Range-finder

As stated previously, an ultrasonic sensor was used in earlier prototypes, but that has been set aside for an infrared range-finder which operates at lower power, with less error. The specifications of the range-finder are listed in table 5.6.

Table 5.6: Technical Information for the infrared range-finder.

Device name	GP2D12
Manufacturer	Sharp
Operating voltage	4.5 V - 5.5 V
Supply current	50 mA max
Output voltage	0.25 V - 2.5 V
Detecting distance	10 cm - 80 cm
Package size	4.06m x 1.5mm x 3.05mm

The purpose of this sensor is to detect the presence of another car ahead on the track. While not implemented in this controller, it is used in the adaptive cruise algorithm designed in [7]. The output of this sensor is an analog signal, and is passed to the A/D converter as described prior.

5.4 PIC Processor

In order to avoid slowing the DSP with the routine functions of updating the servo and motor control signals, a PIC16F874 processor from Microchip has been introduced. Table 5.7 gives a summary of the specifications of the processor. As mentioned in the DSP section,

Table 5.7: Technical Information for the PIC16F874 processor.

Device name	PIC16F874-20/P
Manufacturer	Microchip
Operating voltage	2.0 V - 5.5 V
Operating frequency	DC - 20 MHz
Interrupts	14
I/O Ports	5
Timers	3
Capture/Compare modules	2
Instruction set	35 instructions
Package Type	40-pin DIP

the PIC receives 4 command words from the DSP. At the same time, the PIC sends the current velocity of the vehicle back to the DSP. The PIC has two main functions: speed control and servo control.

5.4.1 Speed Control

When the PIC receives the speed command word from the DSP, it loads that word into two registers. The reason for this is because the PIC is an 8-bit processor and, as stated above, the PWM generator is a 10-bit number. As a result, the 2 LSBs are stored in a separate register from the other 8. The PWM generator is operated at 2kHz, which is required by the h-bridge.

5.4.2 Servo Control

Similar to the speed control, the PIC receives the servo command in two pieces. Once both pieces are received, they are placed together and help until the next update of the servo. The servo operates on a PWM at a frequency of approximately 70Hz. Furthermore, the high time for the servo, from full left to full right, is restricted to 1.5 to 2.5 ms. This is a very specific type of control pulse specifically designed for RC servos.

Chapter 6

Conclusions and Future Work

This chapter summarizes the topic covered in this body of work. Furthermore, it gives the authors thoughts of the results, as well as possible avenues that may be pursued in the future.

6.1 Conclusions

The intent of this thesis has not been to create a control algorithm to keep a vehicle following a path. Rather, it was the intent to augment the controller currently used on the FLASH project to enhance its performance. With this in mind, the known kinematic model was used in conjunction with the dynamic model to create a more precise control algorithm. Once this was done, the previously derived controller was appended to include the dynamics of the system into its derivation. Once the model and controllers were derived, that information was ported into *MatLab* to create a simulation to test the theory.

While there was no real speed controller to compare to the algorithm derived here, it is significant to point out that the current standards for cruise-control systems have windows on the order of 10 to 15% of the nominal speed. The controller derived here, while specific to the FLASH car, can be adjusted to control a full-scale vehicle simply by changing the values for the properties of the vehicle, such as mass, and defining the appropriate equation for the driving force. In terms of practicality, the author sees no reason why the truncated controller should not be used, since the results are effectively identical, although work would need to be done, prior to its use, in terms of the stability of the truncated controller.

It is important to note how easily the velocity is affected by the steering of the vehicle. In real-world applications, the settling time of the controller might want to be monitored more closely because the accelerations present on this small scale would, most likely, be uncomfortable for a passenger in the vehicle. It has been made apparent that the inclusion of the dynamics of the steering mechanism is very important. The only time the steering dynamics would not need to be taken into account is if the time constant approaches half a second, which, for the scale vehicle, is not realistic. Further, while it seems quite obvious, the more often a system can update its information, the better, more efficient, the response. Overall, the design was a success, leading to a set of controllers that allows for a higher degree of accuracy.

At the time of this writing, the hardware application of the algorithms designed here was not possible. The DSP described has not yet been implemented on the vehicle, and the current DSP does not have the memory required to run the control program. Even so, the simulations suggest that the differences in the kinematic control and the dynamic control are quite dramatic. Further, the simulations show that implementing the dynamic steering controller would improve the stability of the vehicle, keeping a tighter handle on the center of the road, by as much as 50%. While in the scale model this is a difference of millimeters, a passenger in a full-scale vehicle would most definitely feel more comfortable and secure if the vehicle swayed less.

6.2 Future Work

Combining this work with the work of others from the FLASH project, such as [6] and [7] might lead some to believe there is nothing left to research. On the other hand, every time a new paper is completed, it leads to new questions. Prior to this thesis, all the research on the FLASH project had been focused around a kinematic controller. Many of the papers created before this mention that the effects of the dynamics are unknown. With the dynamics of the system now defined, better controllers can be derived around those dynamics.

The velocity controller designed here uses motor braking as its only means to slow down, or stop, in a timely manner. Since motor breaking is hard on diesel engines, and nearly impossible on gasoline engines, it stands to reason that the introduction of a braking system outside the motor is necessary. It would further improve the response of an electric motor, and be the only viable way to control a standard car with such accuracy.

One step that seems to be lacking, currently, is one that would improve the possibility of these algorithms making it to consumer automobiles in a shorter amount of time. That topic is adaptability. Currently, all the controllers, at least known to this author, do not adjust the gains or the assumed constants of the system. When it comes down to it, automobile manufacturers are not going to be able to input the exact weight or moments of inertia of a given vehicle. Furthermore, if a new controller has to be designed for each vehicle, the technology will be highly expensive, thus keeping it from reaching the mass market.

To this point, most controllers, and specifically the ones used on the FLASH project, are designed to operate independently, with no interaction for sources outside the vehicle. While this is fine for most highway situations, this would never work in an environment outside the highway. Further, there are situations that can be visualized on the highway where this lack of interaction could have negative effects. If attempts were made to integrate communication, at least to the vehicles immediately around the car in question, into the controllers, then it would be possible to bring this automation into the cities where the traffic is the worst. It could also keep highways moving more smoothly in the event that traffic started to get heavy, in Washington DC for example.

The world of ITS is just getting started. The technology exists to make it a reality. Its main hurdle is the public. Without the support and interest from the public, ITS will continue to be stunted. It is the focus of the FLASH project, not only to give testing grounds to new strategies, but to also prepare the public for the future of driving on public highways. When "Joe Average" becomes comfortable with the idea that the car can drive itself, then ITS will take off. Of course, this will lead to new controllers being designed: the ones that have the ability to ignore input from nervous drivers in emergency situations.

Bibliography

[1] J. Hilton, U. Shankar. *2001 Motor Vehicle Traffic Crashes Injury and Fatality Estimates Early Assessment*. DOT HS 809 439. U.S. Department of Transportation, National Highway Traffic Saftey Administration, National Center for Statistics and Analysis, Washington, DC. April 2002.

[2] A. De Luca, G. Oriolo, C. Samson. *Robot Motion Planning and Control*. Chapter 4. http://www.laas.fr/ jpl/book.html. Dec. 1999.

[3] H. K. Khalil. *Nonlinear Systems, 2nd Edition*. Prentice Hall, Upper Saddle River, NJ, 1996.

[4] R. Dorf and R. Bishop. *Modern Control Systems, 9th Edition*. Prentice Hall, Upper Saddle River, NJ, 2001.

[5] P. Kachroo. *Nonlinear Control Strategies and Vehicle Traction Control*. Dissertation. University of California, Berkeley, 1993.

[6] P. Mellodge. *Feedback Control for a Path Following Robotic Car*. Thesis. Virginia Polytechnic Institute and State University, 2002.

[7] R. D. Henry. *Automatic Ultrasonic Headway Control for a Scaled Robotic Car*. Thesis. Virginia Polytechnic Institute and State University, 2001.

[8] FLASH Project working notes. Virginia Tech Transportation Institute. May 2002.

[9] B. Beham. FLASH notes. Center of Transportation Research. 1995.

[10] M. Bellis. *The History of the Automobile*.
http://inventors.about.com/library/weekly/aacarssteama.htm. 2002.

[11] Antique Automobile Club of America. *Automotive History - A Chronological History*. http://www.aaca.org/history/ 2002.

[12] J. Guldner and V. Utkin. *Stabilization of Non-Holonomic Mobile Robots using Lyapunov Functions for Navigation and Sliding Mode Control*. IEEE Conference on Decision and Control. December 1994.

[13] K. Park, et al. *Point Stabilization of Mobile Robots Via State Space Exact Feedback Linearization.* IEEE Conference on Robotics and Automation. May 1999.

[14] S. Lee, et al. *Control of Car-like Mobile Robots for Posture Stabilization.* IEEE Conference on Intelligent Robots and Systems. 1999.

[15] C. Samson. *Control of Chained Systems Application to Path Following and Time-Varying Point-Stabilization of Mobile Robots.* IEEE Tran. on Automatic Control. Vol. 40, No. 1, January 1995.

[16] C. Shih, et al. *Mixing Input-Output Pseudolinearization and Gain Scheduling Techniques for Stabilization of Mobile Robots.* IEEE Trans. 1996.

[17] G. Walsh, et al. *Stabilization of Trajectories for Systems with Nonholonomic Contstraints.* IEEE Conference on Robotics and Automation. May 1992.

[18] A. Stolsky and X. Hu. *Control of Car-Like Robots Using Sliding Observers for Steering Angle Estimation.* IEEE Conference on Decision and Control. December 1997.

[19] R. M. Desantis. *Path-Tracking for Car-Like Robots with Single and Double Steering.* IEEE Tran. Veh. Technol. Vol. 4, No. 2, May 1995.

[20] M. Egerstedt, et al. *Control of a Car-Like Robot Using a Dynamic Model.* IEEE Conference on Robotics and Automation. May 1998.

[21] J. Baltes and R. Otte. *A Fuzzy Logic Controller for Car-Like Mobile Robots.* IEEE 1999.

[22] M. J. Embrechts, et al. *Fuzzy Logic and Neural Net Control for the "Smarter Car".* IEEE 1995.

[23] R. Fierro and F. L. Lewis. *Practical Point Stabilization of a Nonholonomic Mobile Robot Using Neural Networks.* IEEE Conference on Decision and Control. December 1996.

[24] P. Garnier, et al. *An Hybrid Motion Controller for a Real Car-Like Robot Evolving in a Multi-Vehicle Environment.* Grenoble Cedex 1, France.

[25] W. Oelen, et al. *Implementation of a Hybrid Stabilizing Controller on a Mobile Robot with Two Degrees of Freedom.* IEEE 1994.

Appendix A

Derivative of η

$\dot{\eta} =$

```
(l^2*m+J_eq*tan(phi)^2)/l^2*(2*v_u*tan(phi)*J_eq*l*cos(theta)^3/(1-d*c)^3
*(-k_1*abs(cos(theta)*v_u/(1-d*c))*d-k_2*cos(theta)*v_u*tan(theta)-k_3*abs(cos(theta)
*v_u/(1-d*c))*(-d_c*d*tan(theta)-c*(1-d*c)*(1+sin(theta)^2)/cos(theta)^2+(1-d*c)^2
*tan(phi)/l/cos(theta)^3)-((-(1-d*c)*(1+sin(theta)^2)/cos(theta)^2+c*d
*(1+sin(theta)^2)/cos(theta)^2-2*(1-d*c)*tan(phi)/l/cos(theta)^3*d)*d_c-d*tan(theta)
*dd_c+(-d_c*tan(theta)+c^2*(1+sin(theta)^2)/cos(theta)^2-2*(1-d*c)
*tan(phi)/l/cos(theta)^3*c) *(1-d*c)*tan(theta)+(-d_c*d*(1+tan(theta)^2)-2*c*(1-d*c)
*sin(theta)/cos(theta)-2*c*(1-d*c) *(1+sin(theta)^2)/cos(theta)^3*sin(theta)+3
*(1-d*c)^2*tan(phi)/l/cos(theta)^4*sin(theta)) *(tan(phi)*(1-d*c)/l/cos(theta)-c))
*cos(theta)/(1-d*c)*v_u)/(l^2*m+J_eq*tan(phi)^2)*c+v_u*tan(phi)*J_eq*l
*cos(theta)^3/(1-d*c)^2*(-k_1* sign (cos(theta)*v_u/(1-d*c))*cos(theta)
*v_u/(1-d*c)^2*c*d-k_1*abs(cos(theta)*v_u/(1-d*c))-k_3*sign(cos(theta)*v_u/(1-d*c))
*cos(theta)*v_u/(1-d*c)^2*c*(-d_c*d*tan(theta)-c*(1-d*c)*(1+sin(theta)^2)
/cos(theta)^2+(1-d*c)^2*tan(phi)/l/cos(theta)^3)-k_3*abs(cos(theta)*v_u/(1-d*c))
*(-d_c*tan(theta)+c^2*(1+sin(theta)^2)/cos(theta)^2-2*(1-d*c)*tan(phi)/l
/cos(theta)^3*c)-((2*c*(1+sin(theta)^2)/cos(theta)^2+2*c*tan(phi)/l/cos(theta)^3
*d-2*(1-d*c)*tan(phi)/l/cos(theta)^3)*d_c-tan(theta)*dd_c+2*c^2*tan(phi)/l
/cos(theta)^3*(1-d*c)*tan(theta)-(-d_c*tan(theta)+c^2*(1+sin(theta)^2)/cos(theta)^2
-2*(1-d*c)*tan(phi)/l/cos(theta)^3*c)*c*tan(theta)+(-d_c*(1+tan(theta)^2)+2*c^2
*sin(theta)/cos(theta)+2*c^2*(1+sin(theta)^2)/cos(theta)^3*sin(theta)-6*(1-d*c)
*tan(phi)/l/cos(theta)^4*sin(theta)*c)*(tan(phi)*(1-d*c)/l/cos(theta)-c)
-(-d_c*d*(1+tan(theta)^2)-2*c*(1-d*c)*sin(theta)/cos(theta)-2*c*(1-d*c)
*(1+sin(theta)^2)/cos(theta)^3*sin(theta)+3*(1-d*c)^2*tan(phi)/l/cos(theta)^4
*sin(theta))*tan(phi)*c/l/cos(theta))*cos(theta)/(1-d*c)*v_u-((-(1-d*c)
*(1+sin(theta)^2)/cos(theta)^2+c*d*(1+sin(theta)^2)/cos(theta)^2-2*(1-d*c)
*tan(phi)/l/cos(theta)^3*d)*d_c-d*tan(theta)*dd_c+(-d_c*tan(theta)+c^2
*(1+sin(theta)^2)/cos(theta)^2-2*(1-d*c)*tan(phi)/l/cos(theta)^3*c)*(1-d*c)
*tan(theta)+(-d_c*d*(1+tan(theta)^2)-2*c*(1-d*c)*sin(theta)/cos(theta)-2*c*(1-d*c)
*(1+sin(theta)^2)/cos(theta)^3*sin(theta)+3*(1-d*c)^2*tan(phi)/l/cos(theta)^4
*sin(theta))*(tan(phi)*(1-d*c)/l/cos(theta)-c))*cos(theta)/(1-d*c)^2*v_u*c)
/(l^2*m+J_eq*tan(phi)^2))*d_d+(l^2*m+J_eq*tan(phi)^2)/l^2*(-3*v_u*tan(phi)*J_eq*l
*cos(theta)^2/(1-d*c)^2*(-k_1*abs(cos(theta)*v_u/(1-d*c))*d-k_2*cos(theta)*v_u
*tan(theta)-k_3*abs(cos(theta)*v_u/(1-d*c))*(-d_c*d*tan(theta)-c*(1-d*c)
*(1+sin(theta)^2)/cos(theta)^2+(1-d*c)^2*tan(phi)/l/cos(theta)^3)-((-(1-d*c)
*(1+sin(theta)^2)/cos(theta)^2+c*d*(1+sin(theta)^2)/cos(theta)^2-2*(1-d*c)
*tan(phi)/l/cos(theta)^3*d)*d_c-d*tan(theta)*dd_c+(-d_c*tan(theta)+c^2
*(1+sin(theta)^2)/cos(theta)^2-2*(1-d*c)*tan(phi)/l/cos(theta)^3*c)*(1-d*c)
*tan(theta)+(-d_c*d*(1+tan(theta)^2)-2*c*(1-d*c)*sin(theta)/cos(theta)-2*c*(1-d*c)
*(1+sin(theta)^2)/cos(theta)^3*sin(theta)+3*(1-d*c)^2*tan(phi)/l/cos(theta)^4
*sin(theta))*(tan(phi)*(1-d*c)/l/cos(theta)-c))*cos(theta)/(1-d*c)*v_u)/(l^2*m
+J_eq*tan(phi)^2)*sin(theta)+v_u*tan(phi)*J_eq*l*cos(theta)^3/(1-d*c)^2*(k_1
*sign(cos(theta)*v_u/(1-d*c))*sin(theta)*v_u/(1-d*c)*d+k_2*sin(theta)*v_u*tan(theta)
-k_2*cos(theta)*v_u*(1+tan(theta)^2)+k_3* sign(cos(theta)*v_u/(1-d*c))*sin(theta)
*v_u/(1-d*c)*(-d_c*d*tan(theta)-c*(1-d*c)*(1+sin(theta)^2)/cos(theta)^2+(1-d*c)^2
*tan(phi)/l/cos(theta)^3)-k_3*abs(cos(theta)*v_u/(1-d*c))*(-d_c*d*(1+tan(theta)^2)
-2*c*(1-d*c)*sin(theta)/cos(theta)-2*c*(1-d*c)*(1+sin(theta)^2)/cos(theta)^3
*sin(theta)+3*(1-d*c)^2*tan(phi)/l/cos(theta)^4*sin(theta))-((-2*(1-d*c)*sin(theta)
/cos(theta)-2*(1-d*c)*(1+sin(theta)^2)/cos(theta)^3*sin(theta)+2*c*d*sin(theta)
/cos(theta)+2*c*d*(1+sin(theta)^2)/cos(theta)^3*sin(theta)-6*(1-d*c)*tan(phi)/l
/cos(theta)^4*d*sin(theta))*d_c-d*(1+tan(theta)^2)*dd_c+(-d_c*(1+tan(theta)^2)
+2*c^2*sin(theta)/cos(theta)+2*c^2*(1+sin(theta)^2)/cos(theta)^3*sin(theta)-6*(1-d*c)
*tan(phi)/l/cos(theta)^4*sin(theta)*c)*(1-d*c)*tan(theta)+(-d_c*tan(theta)+c^2
*(1+sin(theta)^2)/cos(theta)^2-2*(1-d*c)*tan(phi)/l/cos(theta)^3*c)*(1-d*c)
*(1+tan(theta)^2)+(-2*d_c*d*tan(theta)*(1+tan(theta)^2)-2*c*(1-d*c)-6*c*(1-d*c)
*sin(theta)^2/cos(theta)^2-6*c*(1-d*c)*(1+sin(theta)^2)/cos(theta)^4*sin(theta)^2-2*c
*(1-d*c)*(1+sin(theta)^2)/cos(theta)^2+12*(1-d*c)^2*tan(phi)/l/cos(theta)^5
*sin(theta)^2+3*(1-d*c)^2*tan(phi)/l/cos(theta)^3)*(tan(phi)*(1-d*c)/l/cos(theta)-c)
+(-d_c*d*(1+tan(theta)^2)-2*c*(1-d*c)*sin(theta)/cos(theta)-2*c*(1-d*c)
*(1+sin(theta)^2)/cos(theta)^3*sin(theta)+3*(1-d*c)^2*tan(phi)/l/cos(theta)^4
*sin(theta))*tan(phi)*(1-d*c)/l/cos(theta)^2*sin(theta))*cos(theta)/(1-d*c)*v_u
+((-(1-d*c)*(1+sin(theta)^2)/cos(theta)^2+c*d*(1+sin(theta)^2)/cos(theta)^2-2
*(1-d*c)*tan(phi)/l/cos(theta)^3*d)*d_c-d*tan(theta)*dd_c+(-d_c*tan(theta)+c^2
*(1+sin(theta)^2)/cos(theta)^2-2*(1-d*c)*tan(phi)/l/cos(theta)^3*c)*(1-d*c)*tan(theta)
+(-d_c*d*(1+tan(theta)^2)-2*c*(1-d*c)*sin(theta)/cos(theta)-2*c*(1-d*c)
*(1+sin(theta)^2)/cos(theta)^3*sin(theta)+3*(1-d*c)^2*tan(phi)/l/cos(theta)^4
*sin(theta))*(tan(phi)*(1-d*c)/l/cos(theta)-c))*sin(theta)/(1-d*c)*v_u)/(l^2*m
+J_eq*tan(phi)^2))*d_theta+(2*J_eq*tan(phi)*(1+tan(phi)^2)/l^2*(-k_v1*v_u+v_u*tan(phi)
*J_eq*l*cos(theta)^3/(1-d*c)^2*(-k_1*abs(cos(theta)*v_u/(1-d*c))*d-k_2*cos(theta)
*v_u*tan(theta)-k_3*abs(cos(theta)*v_u/(1-d*c))*(-d_c*d*tan(theta)-c*(1-d*c)
*(1+sin(theta)^2)/cos(theta)^2+(1-d*c)^2*tan(phi)/l/cos(theta)^3)-((-(1-d*c)
*(1+sin(theta)^2)/cos(theta)^2+c*d*(1+sin(theta)^2)/cos(theta)^2-2*(1-d*c)*tan(phi)/l
/cos(theta)^3*d)*d_c-d*tan(theta)*dd_c+(-d_c*tan(theta)+c^2*(1+sin(theta)^2)
/cos(theta)^2-2*(1-d*c)*tan(phi)/l/cos(theta)^3*c)*(1-d*c)*tan(theta)+(-d_c*d
*(1+tan(theta)^2)-2*c*(1-d*c)*sin(theta)/cos(theta)-2*c*(1-d*c)*(1+sin(theta)^2)
/cos(theta)^3*sin(theta)+3*(1-d*c)^2*tan(phi)/l/cos(theta)^4*sin(theta))*(tan(phi)
*(1-d*c)/l/cos(theta)-c))*cos(theta)/(1-d*c)*v_u)/(l^2*m+J_eq*tan(phi)^2))+(l^2*m
+J_eq*tan(phi)^2)/l^2*(v_u*(1+tan(phi)^2)*J_eq*l*cos(theta)^3/(1-d*c)^2*(-k_1
*abs(cos(theta)*v_u/(1-d*c))*d-k_2*cos(theta)*v_u*tan(theta)-k_3*abs(cos(theta)
*v_u/(1-d*c))*(-d_c*d*tan(theta)-c*(1-d*c)*(1+sin(theta)^2)/cos(theta)^2+(1-d*c)^2
*tan(phi)/l/cos(theta)^3)-((-(1-d*c)*(1+sin(theta)^2)/cos(theta)^2+c*d
*(1+sin(theta)^2)/cos(theta)^2-2*(1-d*c)*tan(phi)/l/cos(theta)^3*d)*d_c-d*tan(theta)
*dd_c+(-d_c*tan(theta)+c^2*(1+sin(theta)^2)/cos(theta)^2-2*(1-d*c)*tan(phi)/l
/cos(theta)^3*c)*(1-d*c)*tan(theta)+(-d_c*d*(1+tan(theta)^2)-2*c*(1-d*c)*sin(theta)
/cos(theta)-2*c*(1-d*c)*(1+sin(theta)^2)/cos(theta)^3*sin(theta)+3*(1-d*c)^2
*tan(phi)/l/cos(theta)^4*sin(theta))*(tan(phi)*(1-d*c)/l/cos(theta)-c))*cos(theta)
/(1-d*c)*v_u)/(l^2*m+J_eq*tan(phi)^2)+v_u*tan(phi)*J_eq*l*cos(theta)^3/(1-d*c)^2
*(-k_3*abs(cos(theta)*v_u/(1-d*c))*(1-d*c)^2*(1+tan(phi)^2)/l/cos(theta)^3
-(-2*(1-d*c)*(1+tan(phi)^2)/l/cos(theta)^3*d*d_c-2*(1-d*c)^2*(1+tan(phi)^2)
/l/cos(theta)^3*c*tan(theta)+3*(1-d*c)^2*(1+tan(phi)^2)/l/cos(theta)^4*sin(theta)
*(tan(phi)*(1-d*c)/l/cos(theta)-c)+(-d_c*d*(1+tan(theta)^2)-2*c*(1-d*c)*sin(theta)
/cos(theta)-2*c*(1-d*c)*(1+sin(theta)^2)/cos(theta)^3*sin(theta)+3*(1-d*c)^2
*tan(phi)/l/cos(theta)^4*sin(theta))*(1+tan(phi)^2)*(1-d*c)/l/cos(theta))
*cos(theta)/(1-d*c)*v_u)/(l^2*m+J_eq*tan(phi)^2)-2*v_u*tan(phi)^2*J_eq^2*l
*cos(theta)^3/(1-d*c)^2*(-k_1*abs(cos(theta)*v_u/(1-d*c))*d-k_2*cos(theta)*v_u
*tan(theta)-k_3*abs(cos(theta)*v_u/(1-d*c))*(-d_c*d*tan(theta)-c*(1-d*c)
*(1+sin(theta)^2)/cos(theta)^2+(1-d*c)^2*tan(phi)/l/cos(theta)^3)-((-(1-d*c)
*(1+sin(theta)^2)/cos(theta)^2+c*d*(1+sin(theta)^2)/cos(theta)^2-2*(1-d*c)
*tan(phi)/l/cos(theta)^3*d)*d_c-d*tan(theta)*dd_c+(-d_c*tan(theta)+c^2
*(1+sin(theta)^2)/cos(theta)^2-2*(1-d*c)*tan(phi)/l/cos(theta)^3*c)*(1-d*c)
*tan(theta)+(-d_c*d*(1+tan(theta)^2)-2*c*(1-d*c)*sin(theta)/cos(theta)-2*c*(1-d*c)
*(1+sin(theta)^2)/cos(theta)^3*sin(theta)+3*(1-d*c)^2*tan(phi)/l/cos(theta)^4
*sin(theta))*(tan(phi)*(1-d*c)/l/cos(theta)-c))*cos(theta)/(1-d*c)*v_u)/(l^2*m
+J_eq*tan(phi)^2)^2*(1+tan(phi)^2)))*v_2+(l^2*m+J_eq*tan(phi)^2)/l^2*(-k_v1
+tan(phi)*J_eq*l*cos(theta)^3/(1-d*c)^2*(-k_1*abs(cos(theta)*v_u/(1-d*c))*d
-k_2*cos(theta)*v_u*tan(theta)-k_3*abs(cos(theta)*v_u/(1-d*c))*(-d_c*d*tan(theta)
-c*(1-d*c)*(1+sin(theta)^2)/cos(theta)^2+(1-d*c)^2*tan(phi)/l/cos(theta)^3)
-((-(1-d*c)*(1+sin(theta)^2)/cos(theta)^2+c*d*(1+sin(theta)^2)/cos(theta)^2
-2*(1-d*c)*tan(phi)/l/cos(theta)^3*d)*d_c-d*tan(theta)*dd_c+(-d_c*tan(theta)+c^2
*(1+sin(theta)^2)/cos(theta)^2-2*(1-d*c)*tan(phi)/l/cos(theta)^3*c)*(1-d*c)
*tan(theta)+(-d_c*d*(1+tan(theta)^2)-2*c*(1-d*c)*sin(theta)/cos(theta)-2*c*(1-d*c)
*(1+sin(theta)^2)/cos(theta)^3*sin(theta)+3*(1-d*c)^2*tan(phi)/l/cos(theta)^4
*sin(theta))*(tan(phi)*(1-d*c)/l/cos(theta)-c))*cos(theta)/(1-d*c)*v_u)/(l^2*m
+J_eq*tan(phi)^2)+v_u*tan(phi)*J_eq*l*cos(theta)^3/(1-d*c)^2*(-k_1* sign (cos(theta)
*v_u/(1-d*c))*cos(theta)/(1-d*c)*d-k_2*cos(theta)*tan(theta)-k_3*sign(cos(theta)
*v_u/(1-d*c))*cos(theta)/(1-d*c)*(-d_c*d*tan(theta)-c*(1-d*c)*(1+sin(theta)^2)
/cos(theta)^2+(1-d*c)^2*tan(phi)/l/cos(theta)^3)-((-(1-d*c)*(1+sin(theta)^2)
/cos(theta)^2+c*d*(1+sin(theta)^2)/cos(theta)^2-2*(1-d*c)*tan(phi)/l/cos(theta)^3*d)
*d_c-d*tan(theta)*dd_c+(-d_c*tan(theta)+c^2*(1+sin(theta)^2)/cos(theta)^2-2*(1-d*c)
*tan(phi)/l/cos(theta)^3*c)*(1-d*c)*tan(theta)+(-d_c*d*(1+tan(theta)^2)-2*c*(1-d*c)
*sin(theta)/cos(theta)-2*c*(1-d*c)*(1+sin(theta)^2)/cos(theta)^3*sin(theta)+3
*(1-d*c)^2*tan(phi)/l/cos(theta)^4*sin(theta))*(tan(phi)*(1-d*c)/l/cos(theta)-c))
*cos(theta)/(1-d*c))/(l^2*m+J_eq*tan(phi)^2))*d_vu+(l^2*m+J_eq*tan(phi)^2)/l^2
*(2*v_u*tan(phi)*J_eq*l*cos(theta)^3/(1-d*c)^3*(-k_1*abs(cos(theta)*v_u/(1-d*c))
*d-k_2*cos(theta)*v_u*tan(theta)-k_3*abs(cos(theta)*v_u/(1-d*c))*(-d_c*d*tan(theta)
-c*(1-d*c)*(1+sin(theta)^2)/cos(theta)^2+(1-d*c)^2*tan(phi)/l/cos(theta)^3)
-((-(1-d*c)*(1+sin(theta)^2)/cos(theta)^2+c*d*(1+sin(theta)^2)/cos(theta)^2-2
*(1-d*c)*tan(phi)/l/cos(theta)^3*d)*d_c-d*tan(theta)*dd_c+(-d_c*tan(theta)+c^2
*(1+sin(theta)^2)/cos(theta)^2-2*(1-d*c)*tan(phi)/l/cos(theta)^3*c)*(1-d*c)
*tan(theta)+(-d_c*d*(1+tan(theta)^2)-2*c*(1-d*c)*sin(theta)/cos(theta)-2*c*(1-d*c)
*(1+sin(theta)^2)/cos(theta)^3*sin(theta)+3*(1-d*c)^2*tan(phi)/l/cos(theta)^4
*sin(theta))*(tan(phi)*(1-d*c)/l/cos(theta)-c))*cos(theta)/(1-d*c)*v_u)/(l^2*m
+J_eq*tan(phi)^2)*d+v_u*tan(phi)*J_eq*l*cos(theta)^3/(1-d*c)^2*(-k_1*
sign (cos(theta)*v_u/(1-d*c))*cos(theta)*v_u/(1-d*c)^2*d^2-k_3*sign(cos(theta)
*v_u/(1-d*c))*cos(theta)*v_u/(1-d*c)^2*d*(-d_c*d*tan(theta)-c*(1-d*c)
```

```
*(1+sin(theta)^2)/cos(theta)^2+(1-d*c)^2*tan(phi)/l/cos(theta)^3)-k_3*abs(cos(theta)
*v_u/(1-d*c))*(-(1-d*c)*(1+sin(theta)^2)/cos(theta)^2+c*d*(1+sin(theta)^2)
/cos(theta)^2-2*(1-d*c)*tan(phi)/l/cos(theta)^3*d)-((2*d*(1+sin(theta)^2)
/cos(theta)^2+2*d^2*tan(phi)/l/cos(theta)^3)*d_c+(2*c*(1+sin(theta)^2)/cos(theta)^2
+2*c*tan(phi)/l/cos(theta)^3*d-2*(1-d*c)*tan(phi)/l/cos(theta)^3)*(1-d*c)*tan(theta)
-(-d_c*tan(theta)+c^2*(1+sin(theta)^2)/cos(theta)^2-2*(1-d*c)*tan(phi)/l
/cos(theta)^3*c)*d*tan(theta)+(-2*(1-d*c)*sin(theta)/cos(theta)-2*(1-d*c)
*(1+sin(theta)^2)/cos(theta)^3*sin(theta)+2*c*d*sin(theta)/cos(theta)+2*c*d
*(1+sin(theta)^2)/cos(theta)^3*sin(theta)-6*(1-d*c)*tan(phi)/l/cos(theta)^4
*d*sin(theta))*(tan(phi)*(1-d*c)/l/cos(theta)-c)+(-d_c*d*(1+tan(theta)^2)
-2*c*(1-d*c)*sin(theta)/cos(theta)-2*c*(1-d*c)*(1+sin(theta)^2)/cos(theta)^3
*sin(theta)+3*(1-d*c)^2*tan(phi)/l/cos(theta)^4*sin(theta))*(-tan(phi)
*d/l/cos(theta)-1))*cos(theta)/(1-d*c)*v_u-((-(1-d*c)*(1+sin(theta)^2)/cos(theta)^2
+c*d*(1+sin(theta)^2)/cos(theta)^2-2*(1-d*c)*tan(phi)/l/cos(theta)^3*d)*d_c
-d*tan(theta)*dd_c+(-d_c*tan(theta)+c^2*(1+sin(theta)^2)/cos(theta)^2-2*(1-d*c)
*tan(phi)/l/cos(theta)^3*c)*(1-d*c)*tan(theta)+(-d_c*d*(1+tan(theta)^2)-2*c*(1-d*c)
*sin(theta)/cos(theta)-2*c*(1-d*c)*(1+sin(theta)^2)/cos(theta)^3*sin(theta)+3
*(1-d*c)^2*tan(phi)/l/cos(theta)^4*sin(theta))*(tan(phi)*(1-d*c)/l/cos(theta)-c))
*cos(theta)/(1-d*c)^2*v_u*d)/(l^2*m+J_eq*tan(phi)^2))*d_c*d_s+1/l*v_u*tan(phi)
*J_eq*cos(theta)^3/(1-d*c)^2*(k_3*abs(cos(theta)*v_u/(1-d*c))*d*tan(theta)-(-(1-d*c)
*(1+sin(theta)^2)/cos(theta)^2+c*d*(1+sin(theta)^2)/cos(theta)^2-2*(1-d*c)*tan(phi)
/l/cos(theta)^3*d-tan(theta)^2*(1-d*c)-d*(1+tan(theta)^2)*(tan(phi)*(1-d*c)/l
/cos(theta)-c))*cos(theta)/(1-d*c)*v_u)*dd_c*d_s+1/l*v_u^2*tan(phi)*J_eq
*cos(theta)^4/(1-d*c)^3*d*tan(theta)*ddd_c*d_s
```

Appendix B

MatLab Files

B.1 Initial.m

```
% This subprogram initializes the car and inputs

global theta0;


%%%%%%%%%%%%%%%%%      Simulation Control Constants   %%%%%%%%%%%%
control = 1; %Chooses which controllers are used
% 0 = Velocity controller without lateral controller
% 1 = Dynamic velocity and lateral controller, THIS IS THE FULL CONTROLLER
% 2 = Truncated velocity controller and Lateral controller
% 3 = No dynamic controllers, just kinematic controllers

plot_me = 1;     %Chooses what plots to show
% 0 = No plots
% 1 = plots all data for control=1 - must be Full Controller
% 2 = plots all data for control~=1 - does not work for Full Controller
% 3 = plots comparisons of kinematic controllers to dynamic controllers
% run the full controller second
show_me = 1; %Shows the movie 0=off 1=on



%Control gains
k1 = 2000; % The first three are for the lateral controller
k2 = 100;
k3 = 100;
kv1 = 5; % The last two are for the speed controller
kv2 = 1250;
```

```
% Define's the path
x_path=[-2.5:.01:2.5];
[y_path,path_fun]=F_path(x_path);


% Initial conditions (must be on the road!!!)
init_x = -1.14;
x0 = init_x; % position of center of rear wheels along the x axis (m)
dx0 = x0+T; % for setting theta0
x = x0;
y0 = eval(path_fun); % position of center of rear wheels along the y axis (m)
x = dx0;
dy0 = eval(path_fun); % for setting theta0
theta0 = atan2(dy0-y0,dx0-x0); % body angle (rad)
phi0 = 0*pi/180; % steering angle (limited -45 to 45 degrees) (rad)

vu0 = .5;
Fd0 = 0;
vd = 1.0; %desired velocity, in m/s actual speed (not scale)
d_vd = 0; %dvd = diff(vd);


mph = vd*0.62137*36 %desired full scale speed
u2 = 0; % v2 transformed (rad/s)


%Sets the scene and viewpoint of the movie
if show_me == 1
   axis equal;
   grid on;axis([x0-2*W -x0+2*W y0-l -y0+l]);
   %axis off;
   view(2);
   %view(3);
end



%initializing other variables
theta0_prev = theta0;
phi0_prev = phi0;

PHI = phi0;
FD = Fd0;
c = phi0;

prev_front = 0;
prev_back = 0;

% create car and locate it on the road

if show_me == 1
```

```
   hold on
   car = CreateCar(W,l,H,D,F,R,b,'green');
   locate(car,[x0 y0 0]);
   aim(car,[x0+cos(theta0) y0+sin(theta0) 0]);
   turn(car.tire_fl,'z',phi0*180/pi);
   turn(car.tire_fr,'z',phi0*180/pi);
end
```

B.2 constants.m

```
% Constants for the car

l = 10 *2.54/100; % distance between rear wheel and front wheel (m)
W = 6.5 *2.54/100; % space between wheels (m)
H = 6 *2.54/100/2; % height (m)
D = 2.5 *2.54/100; % diameter of wheels (m)
F = 1.25 *2.54/100; % width of front wheels (m)
R = 2 *2.54/100; % width of rear wheels (m)
B_w = 5 *2.54/100; % width of bumper
T = 0.01; % sampling time (s)
b = 3.5 *2.54/100; % distance from CG to rear axle
a = l-b; % distance from CG to front axle
sensors = 16; % number of sensors in a bumper

Ra = 1.9;
La = 1.064e-4;
Nw = 81;
Nm = 21;
Rw = 31.75e-3;
bm = 3.397e-5;
Km = 6.7831e-3;
Kb = Km;
m = 1.4175;
J = W*l*m;
Jeq = b^2*m+J;
```

B.3 f-path.m

```
function [y,funct] = F_path( x_list );

funct = 'x^3';  % only insert the right side of the equation

for jj=1:length(x_list)
   x=x_list(jj);
   y(jj) = eval(funct);
end
```

B.4 derr.m

```
function df= derr(f,x,dx);

df=jacobian(f,x)*dx;

df = char(df);

spell_check = 1;
while spell_check < length(df)-5
if df(spell_check:spell_check+5)=='signum'
df(spell_check:spell_check+5)=' sign ';
end
spell_check = spell_check+1;
end

df = sym(df);
```

B.5 model2.m

```
% Full Dynamic Feedback control model
%clear all;close all;
hold on;

%Define Sampling time of sensors
global T;
T = 1; %this is only here to allow T a value
%Creating symbolic variables
syms s th b l phi0 m J vu0 Jeq d c c1 c2 c3 u1 u2 w1 w2 v1 v2 real
syms Fd0 Ra La bm Km Kb Nm Nw Rw a1 a2 k1 k2 k3 kv1 kv2 real
syms eta d_s d_d d_th d_vu d_eta vd d_vd d_eta1 real

TIME = [s d th phi0 vu0 Fd0 c c1 c2];
D_TIME = [d_s d_d d_th v2 d_vu w1 c1*d_s c2*d_s c3*d_s]';
ARC = [s c c1 c2]; D_ARC = [1 c1 c2 c3]';

% Adding known constants
constants;

%Creating symbolic equations
GAM = cos(phi0)^2*(l^2*m+Jeq*tan(phi0)^2);
D_S = cos(th)*vu0/(1-d*c);
D_D = sin(th)*vu0;
D_TH = (tan(phi0)/l - c*cos(th)/(1-d*c))*vu0;
D_VU = l^2*cos(phi0)^2/GAM*Fd0 - vu0*tan(phi0)*Jeq*v2/GAM;
CHI2 = d; CHI3 = (1-d*c)*tan(th);
CHI4 = -c1*d*tan(th) - c*(1-c*d)*(1+sin(th)^2)/cos(th)^2
+ (1-d*c)^2*tan(phi0)/l/cos(th)^3;
```

```
A1 = derr(CHI4,ARC,D_ARC) + diff(CHI4,d)*(1-d*c)*tan(th)
+ diff(CHI4,th)*(tan(phi0)*(1-d*c)/l/cos(th)-c);
A2 = l*cos(th)^3*cos(phi0)^2/(1-d*c)^2;
U2 = (18.519*phi0 + v2)/2.327;
W2 = -k1*abs(cos(th)*vu0/(1-d*c))*CHI2 - k2*cos(th)*vu0*CHI3/(1-d*c)
- k3*abs(cos(th)*vu0/(1-d*c))*CHI4;
V2 = A2*(W2 - A1*cos(th)/(1-d*c)*vu0);
ETA1 = GAM/l^2/cos(phi0)^2*(-kv1*vu0 + vu0*tan(phi0)*Jeq*V2/GAM);
D_ETA1 = derr(ETA1,vu0,D_VU);
D_ETA = derr(ETA1,TIME,D_TIME);
ETA = GAM/l^2/cos(phi0)^2*(-kv1*(vu0-vd) - d_vd + vu0*tan(phi0)*Jeq*V2/GAM);
V1 = -vu0*l^2*cos(phi0)^2/GAM - kv2*(Fd0-eta);
W1 = v1 + d_eta;
W1trunc = v1 + d_eta1;
U1 = Rw*Nm*La/Nw/Km*(w1 + (Ra*bm+Km*Kb)*Nw^2/Nm^2/Rw^2*vu0 + Ra/La*Fd0);


%Initializing the path and the car's position on the path, speed, etc
initial;

if show_me == 1
   plot(x_path,y_path);
   title('Traversing of the Path');
end

i = 0;

curv = (diff(path_fun,2))/((1+diff(path_fun)^2)^(3/2));
curv1 = diff(curv);
curv2 = diff(curv1);
curv3 = diff(curv2);
while x0 < -init_x
   i = i+1;

   % actual error
   ef(i)=FindError(x0,y0,theta0,path_fun,l,W);
   eb(i)=FindError(x0,y0,theta0,path_fun,0,W);
   theta_p(i) = FindHeadingAngle(ef(i),eb(i),l);

   % determine array output based on car position
   % discretized error
   front(i) = sensor(ef(i),B_w,prev_front,sensors);
   back(i) = sensor(eb(i),B_w,prev_back,sensors);
   theta_p_hat(i) = FindHeadingAngle(front(i),back(i),l);


   % assign the states
   % curvature

   x=x0;
   CURV(i) = eval(curv);
```

```
CURV1(i) = eval(curv1);
CURV2(i) = eval(curv2);
CURV3(i) = eval(curv3);

c = CURV(i);
c1 = CURV1(i);
c2 = CURV2(i);
c3 = CURV3(i);
% actual error

th = theta_p(i);
d = eb(i);

chi4 = eval(CHI4);
chi3 = eval(CHI3);
chi2 = eval(CHI2);
X2(i) = chi2;
X3(i) = chi3;
X4(i) = chi4;

% discretized error
th = theta_p_hat(i);
d = back(i);

chi4_hat = eval(CHI4);
chi3_hat = eval(CHI3);
chi2_hat = eval(CHI2);
X2_hat(i) = chi2_hat;
X3_hat(i) = chi3_hat;
X4_hat(i) = chi4_hat;

% transform the variables and create steering command
dummy = cos(th)/(1-d*c);
th = theta_p_hat(i); % discretized error
v2 = eval(V2);
u2 = eval(U2);
if control == 0 | control == 3
   u2 = u2-18.519*phi0/2.327;
end

U2out(i) = u2;

Vu(i) = vu0;
V2out(i) = v2;
Vs(i) = vu0/dummy;

% generating velocity command

d_s = eval(D_S);
d_d = eval(D_D);
d_vu = eval(D_VU);
```

```
d_th = eval(D_TH);

eta = eval(ETA);
ETAout(i)=eta;
v1 = eval(V1);
V1out(i) = v1;
d_eta = eval(D_ETA);
DERR_ETA(i) = d_eta;
DERR_ETA1(i) = eval(D_ETA1);
w1 = eval(W1);

W1out(i) = w1;

d_eta1 = eval(D_ETA1);
w1trunc(i) = eval(W1trunc);

if control == 2
    w1 = w1trunc(i);
end

u1 = eval(U1);

if control == 3
   u1 = .5050*vd;
end
U1out(i) = u1;


   % update car dynamics
[t_sim state output] = sim('model_sim',[0 T]); % returns vectors x,y,theta,phi
x0 = x(length(x));
y0 = y(length(y));
theta0 = theta(length(theta));
phi0 = phi(length(phi));
Fd0 = Fd(length(Fd));
vu0 = vu(length(vu));

while theta0 > pi
   theta0 = theta0-2*pi;
end
while theta0 <= -pi
   theta0 = theta0+2*pi;
end
THETA(i) = theta0;
X(i) = x0;
Y(i) = y0;
PHI(i) = phi0;
FD(i) = Fd0;

mph(i) = vu0/1000*0.62137*3600*10;
```

```
   if (show_me == 1)
      % update animation
      locate(car,[x0,y0 0]);
      turn(car,'z',(theta0-theta0_prev)*180/pi);
      turn(car.tire_fl,'z',(phi0-phi0_prev)*180/pi);
      turn(car.tire_fr,'z',(phi0-phi0_prev)*180/pi);
      M(i) = getframe;
   else
      % to show progress and avoid crashes
      x0
   end

   theta0_prev = theta0;
   phi0_prev = phi0;

   prev_front = front(i);
   prev_back = back(i);

end

%---------- code is good to here ---------------
t = [0:T:T*(i-1)];

if control ~= 1
   U1out2=U1out;clear U1out;
   FD2=FD;clear FD;
   Vu2=Vu;clear Vu;
   theta_p2=theta_p;clear theta_p;
   ef2=ef;clear ef;
   eb2=eb;clear eb;
   X_2=X;clear X;
   Y2=Y;clear Y;
   t2=t;clear t;
end


if(show_me == 1)
   plot(X,Y,'g',X+cos(THETA)*l,Y+sin(THETA)*l,'r');
end

hold on;

if(plot_me == 1)

   figure(2);plot(t,U1out);title('Control Effort');
   xlabel('Time (s)');ylabel('Voltage (V)');

   figure(3);plot(t,FD);title('Driving Force');
   xlabel('Time (s)');ylabel('Force (N)');

   figure(4);plot(t,Vu);title('Velocity');
   xlabel('Time (s)');ylabel('Velocity (m/s)');

figure(5)
plot(t,theta_p*180/pi,t,theta_p_hat*180/pi,'r')
   title('Error in Orientation');
   xlabel('Time(S)');ylabel('Error (Degrees)');

figure(6)
subplot(2,1,1),plot(t,ef,t,front,'r')
title('Front Error');ylabel('Error (m)')
subplot(2,1,2),plot(t,eb,t,back,'r')
title('Back Error');xlabel('Time(S)');ylabel('Error (m)')

figure(7)
subplot(3,1,1),plot(t,X2,t,X2_hat,'r')
ylabel('x_2')
subplot(3,1,2),plot(t,X3,t,X3_hat,'r')
ylabel('x_3')
   subplot(3,1,3),plot(t,X4,t,X4_hat,'r')
   xlabel('Time(S)');ylabel('x_4')

figure(8)
subplot(3,1,1),plot(t,CURV);%,t,curv_hat,'r')
title('Curvature of path')
subplot(3,1,2),plot(t,CURV1);%,t,curv_hat1,'r')
   subplot(3,1,3),plot(t,CURV2);%,t,curv_hat2,'r')
   xlabel('Time(S)');

figure(9)
plot(t,X);
title('X');
   xlabel('Time(S)');

figure(10)
plot(t,Y);
   title('Y');
   xlabel('Time(S)');

   maxes = [max(U1out) max(Vu)]
end

if(plot_me == 2)
figure(5)
plot(t2,theta_p2*180/pi,'r')
   title ('Error in Orientation')
   xlabel('Time (s)');ylabel('Car Angle (degrees)');

figure(6)
subplot(2,1,1),plot(t2,ef2,'r')
   title('Error')
   ylabel('Front Error (m)');
```

```
subplot(2,1,2),plot(t2,eb2,'r')
   ylabel('Back Error (m)');
   xlabel('Time (s)');


%figure(7)
%subplot(3,1,1),plot(t2,X2,color)
%ylabel('x_2')
%subplot(3,1,2),plot(t2,X3,color)
%ylabel('x_3')
%subplot(3,1,3),plot(t2,X4,color)
%xlabel('Time(S)');;ylabel('x_4')

figure(8)
subplot(3,1,1),plot(t2,CURV);%,t,curv_hat,'r')
title('Curvature of path')
subplot(3,1,2),plot(t2,CURV1);%,t,curv_hat1,'r')
    subplot(3,1,3),plot(t2,CURV2);%,t,curv_hat2,'r')
    xlabel('Time(S)');%

figure(9)
plot(t2,X_2,'r');
    xlabel('Time(S)');
    title('X');

figure(10)
plot(t2,Y2,'r');
   xlabel('Time(S)');
   title('Y');
end

if plot_me == 3

   figure(2);plot(t,U1out,t2,U1out2,'r');title('Control Effort');
   xlabel('Time (s)');ylabel('Voltage (V)');
   legend('Dynamic Controller','Kinematic Controller');

   figure(3);plot(t,FD,t2,FD2,'r');title('Driving Force');
   xlabel('Time (s)');ylabel('Force (N)');
   legend('Dynamic Controller','Kinematic Controller');

   figure(4);plot(t,Vu,t2,Vu2,'r');title('Velocity');
   xlabel('Time (s)');ylabel('Velocity (m/s)');
   legend('Dynamic Controller','Kinematic Controller');

   figure(5)
plot(X,theta_p*180/pi,X_2,theta_p2*180/pi,'r')
   title('Error in Orientation');
   xlabel('Position along path');ylabel('Error (Degrees)')
   legend('Dynamic Controller','Kinematic Controller');
   axis tight;
```

```
figure(6)
subplot(2,1,1),plot(X,ef,X_2,ef2,'r')
   title('Front Error');ylabel('Error (m)')
   legend('Dynamic Controller','Kinematic Controller');
   axis tight;

subplot(2,1,2),plot(X,eb,X_2,eb2,'r')
   title('Back Error');ylabel('Error (m)')
   xlabel('Position along path');
   legend('Dynamic Controller','Kinematic Controller');
   axis tight;

   figure(7)
   plot(X,abs(ef)+abs(eb),X_2,abs(ef2)+abs(eb2),'r')
   ylabel('Error (m)');
   xlabel('Position along path');axis tight;

end
```

B.6 FindError.m

```
function error = FindError(x0,y0,theta0,funct,dr,W)

%theta0 is the orientation of the vehicle
%funct is the path function
%dr is the length of the vehicle

x1 = x0+dr*cos(theta0);
y1 = y0+dr*sin(theta0);
m = tan(theta0 - pi/2);


path=sprintf('y=%s',funct);
axle=sprintf('y=%d*x+(%d)',m,y1-x1*m);
[x,y]=solve(path,axle);
x=double(x);y=double(y);

dist = sqrt( (x1-x).^2 + (y1-y).^2);
[err,ind] = min(dist);
y2=y(ind);x2=x(ind);

if (theta0 > -pi) & (theta0 <= -pi/2)
   if abs(y1-y2) >= 0.00001
      if y1 >= y2
         error = -err;
      else
         error = err;
      end
```

```
else
    if x1 >= x2
   error = err;
else
    error = -err;
      end
   end
end

if (theta0 > -pi/2) & (theta0 <= 0)
   if abs(y1-y2) >= 0.00001
      if y1 >= y2
         error = err;
      else
         error = -err;
      end
else
    if x1 >= x2
   error = err;
else
    error = -err;
      end
   end
end

if (theta0 > 0) & (theta0 <= pi/2)
   if abs(y1-y2) >= 0.00001
      if y1 >= y2
         error = err;
      else
         error = -err;
      end
else
    if x1 >= x2
   error = -err;
else
    error = err;
      end
   end
end

if (theta0 > pi/2) & (theta0 <= pi)
   if abs(y1-y2) >= 0.00001
      if y1 >= y2
         error = -err;
      else
         error = err;
      end
else
    if x1 >= x2
   error = -err;
```

```
else
    error = err;
      end
   end
end
```

B.7 sensor.m

```
% sensor.m
function error = sensor(d,B_w,p,s)

% IR sensors
prob = 1;
space = B_w/s;


array = ones(s,1);
for k = -0.5*s:0.5*s-1
   if (d >= k*space) & (d <= (k+1)*space)
      array(s/2-k) = 0;
   end
   if rand > prob
      array(s/2-k) = 1-array(s/2-k);
   end
end

error = 0;
num = 0;
val = (s+1)/2;

for k = 1:s
   if array(k) == 0
       num = num+1;
       error = error+(val-k)*space;
   end
end


if num ~= 0
   error = error/num;
else
   error = B_w/2*sign(p);
end
```

B.8 FindHeadingAngle.m

```
% Find the heading angle

function angle = FindHeadingAngle(e_front,e_back,l)

angle = atan((e_front-e_back)/l); % radians
```

Appendix C

FLASH Car Pictures

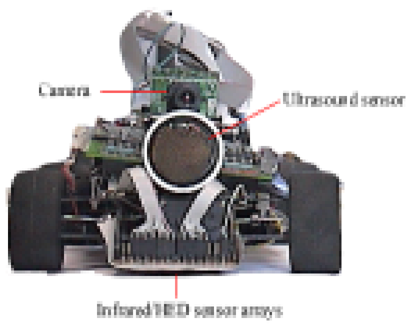

Figure C.1: The first prototype, affectionately named 'Twitchy'

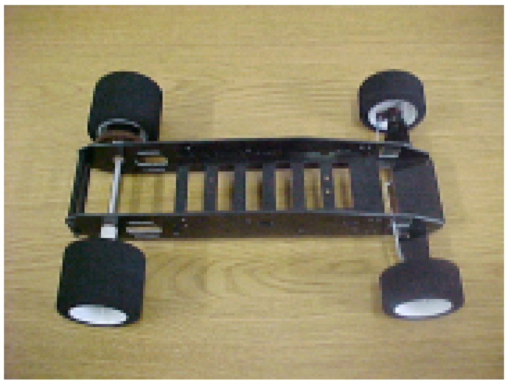

Figure C.2: The chassis on which the everything is build

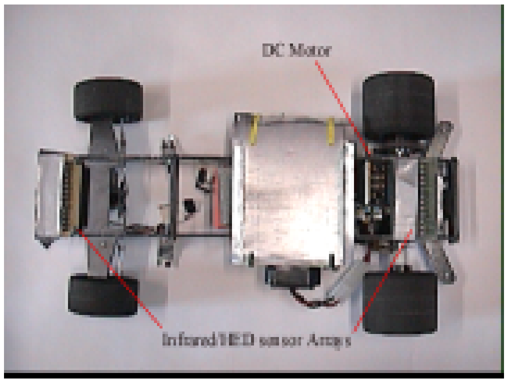

Figure C.3: Underside of the FLASH car

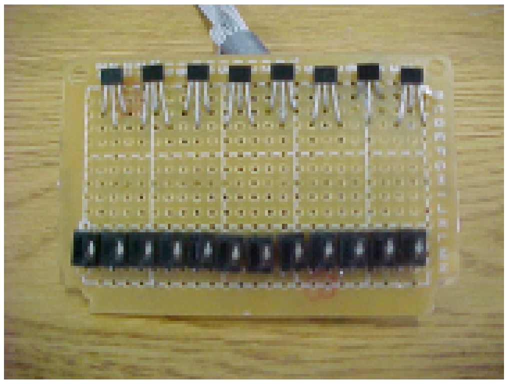

Figure C.4: Closeup of a infrared and magnetic sensor array

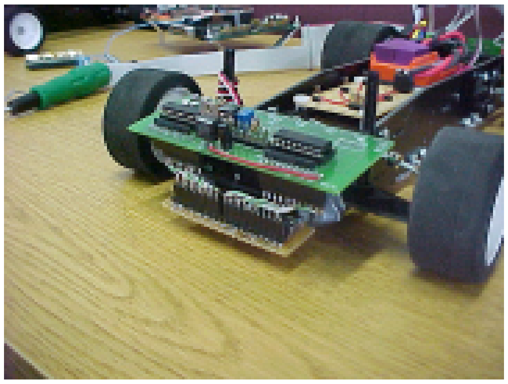

Figure C.5: Display of previous sensor board connected to its logic board

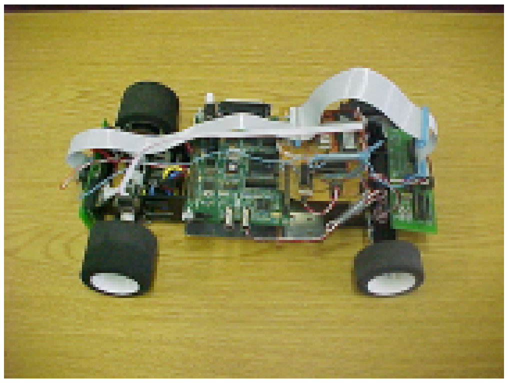
Figure C.6: Fully built autonomous FLASH car

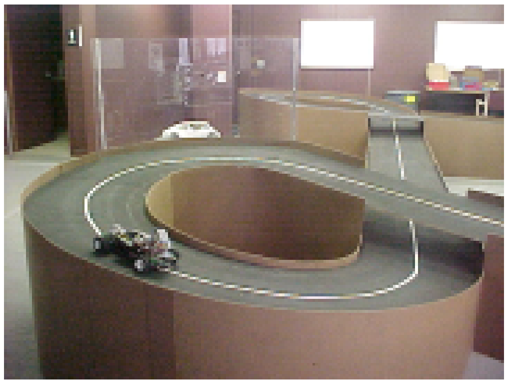
Figure C.7: Single car following the track

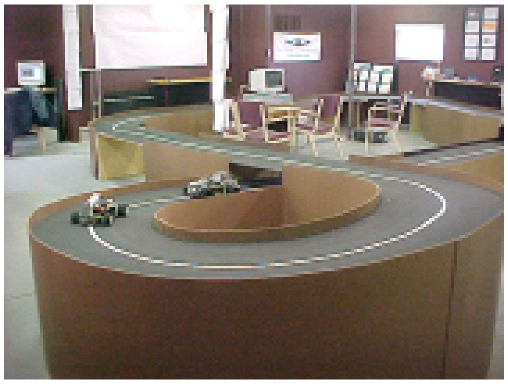
Figure C.8: Example of adaptive cruise control

Vita

Eric Moret was born and raised in Brandon, MS. In 1993, he and his mother moved to Northern VA where he graduated from Lake Braddock Secondary School in 1995. From 1995 to 1998, Eric attended No. VA Community College while trying to decide whether to pursue a career in law enforcement, photography, or engineering. In the fall of 1998, he finally settled on engineering and entered Virginia Tech with sights in a degree in computer engineering. He graduated from Virginia Tech in 2001 with a Bachelor's degree in Mechanical Engineering. His final year in his undergrad he was introduce to control theory and, subsequently, pursued a Master's degree in Mechanical Engineering, to focus in kinematics and controls. Partially through his MSME in 2002, Eric changed graduate majors; this time to pursue a Master's degree in Electrical Engineering. Areas on interest and research include robotics, control theory, prosthetics, bio-mechanical engineering, system theory, and cybernetics.

While at Virginia Tech, Eric found himself involved with the Virginia Tech Fencing Club. He was a member of the Epee team, competing collegiately, as well as individually. He later became President, reaching his pinnacle in that role by hosting the collegiate championships.

More important than any degree or club, Eric found his wife while at Tech. She has stood beside him throughout the writing of this work, and continues to as Eric pursues his next degree.