2012 Intelligent Vehicles Symposium
Alcalá de Henares, Spain, June 3-7, 2012

Up to the Limits: Autonomous Audi TTS

Joseph Funke, Paul Theodosis, Rami Hindiyeh, Ganymed Stanek, Krisada Kritatakirana, Chris Gerdes
Stanford University
Dynamic Design Laboratory
Department of Mechanical Engineering
Stanford, CA 94305, USA
{jfunke, ptheodosis, hindiyeh, ganymed, krisadak, gerdes}@stanford.edu

Dirk Langer, Marcial Hernandez, Bernhard Müller-Bessler, Burkhard Huhnke
Electronics Research Lab
500 Clipper Drive
Belmont, CA 94002, USA
{Dirk.Langer, Marcial.Hernandez, Bernhard.Mueller, Burkhard.Huhnke}@vw.com

***Abstract*— This paper presents a novel approach to autonomous driving at the vehicle's handling limits. Such a system requires a high speed, consistent control signal as well as numerous safety features capable of monitoring and stopping the vehicle. When operating, the system's high level controller utilizes a highly accurate differential GPS and known friction values to drive a precomputed path at the friction limits of the vehicle. The system was tested in a variety of road conditions, including the challenging Pikes Peak Hill climb. Results from this work can be extended to improve driving safety and accident avoidance in vehicles.**

I. INTRODUCTION

Accident prevention and crash avoidance are important topics in the automotive world. Driving safety has improved significantly over the years through the introduction of seat belts in 1959, Anti-Lock Brakes (ABS) in 1971, Air Bags in 1990 and Electronic Stability Control (ESC) in 1995. In the United States alone, however, there are still more than 10 million accidents with more than 30,000 fatalities per year according to NHTSA statistics [6]. To improve driving safety further the next steps need to involve the development of systems that prevent the vehicle from departing its designated travel lane and avoid collisions with other vehicles and pedestrians. Apart from the ability to perceive the vehicle's environment and driving conditions it is also necessary to be able to control the vehicle at its dynamic limits in order to ensure a safe accident avoidance maneuvre. Race car drivers are among the safest drivers, being able to keep their vehicle on an optimal racing trajectory at the friction limits. Rally car drivers in addition must account for many unknowns and trade stability for controllability. Both types of drivers therefore served as role models for the research described in this paper.

As a research vehicle we selected an Audi TTS, equipped with a drive-by-wire system based on an architecture and hardware developed for the DARPA Urban Challenge and previously implemented on our *Junior3* vehicle [5] (see Fig. 1.

Section II discusses the hardware and software architecture that enable high speed autonomous operation and safety. Section III describes how paths are generated. These optimized paths are computed offline and downloaded to the vehicle.

Fig. 1. Audi TTS research vehicle

Section IV introduces the high level control necessary to track these paths at the vehicle's limits. Our final test area was the Pikes Peak Hill Climb course in Colorado, which poses difficult challenges with its combination of paved and dirt roads, significant bank and grade, and narrow road surfaces.

II. SYSTEM ARCHITECTURE

Autonomously driving a vehicle at its handling limits requires a high speed and safety rich software architecture. Our structure has been designed to achieve hard real-time control at 200Hz, the maximum possible with our GPS system, while providing numerous safety features to bring the vehicle to a stop in case of an erroneous condition. The System Overview section provides more detail about the overall setup. The High Speed Real-time Control and Safety Features section discuss how these two design goals are met.

A. System Overview

The system, as outlined in Fig. 2, is structured around two main computers, a High Level Controller and a Monitoring System, which together provide a 200Hz control structure with various safety features. The High Level Controller

978-1-4673-2118-1/$31.00 ©2012 IEEE

acquires vehicle state information from the production vehicle sensors and from an integrated GPS/INS system. Data from vehicle sensors is collected through the vehicle's CAN network, and GPS data is received via UDP/IP. At each time step, the High Level Controller outputs driving commands, such as throttle position and desired road wheel angle, via CAN to the drive-by-wire system. The Monitoring System is connected to the GPS, drive-by-wire system, and vehicle CAN network, allowing it to monitor all of these systems for potential errors.

The remaining key systems are the GPS, wireless stop system, and drive-by-wire system. The GPS is an Applanix POS LV420 system, a differential GPS with an integrated IMU. This system was chosen for its high accuracy and low IMU drift (in case of a GPS outage), as well as its familiarity among members on our team that had used it on Junior, the Stanford/Volkswagen DARPA Urban Challenge entry [3], [5]. A Torc SafeStop stop system is used to provide a wireless remote emergency stop. Finally, the drive-by-wire system consists of a series of PIC33 microcontrollers developed by Volkswagen ERL to interface with the production vehicle actuators [5].

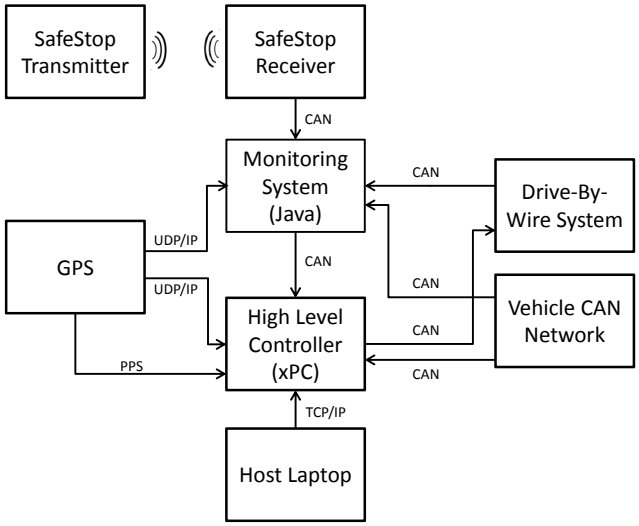

Fig. 2. System Architecture Overview

B. High Speed Real-time Control

The overall system has been designed to ensure a 200Hz, hard real-time control signal. At lower speeds, the vehicle can function adequately while running at slower control rates or with intermittent GPS data. At higher speeds, however, vehicle states rapidly change between time steps and missed or mismatched data (executing with only some of the new GPS information) can be problematic. For example, at 100mph, the vehicle covers about 0.22m of distance each time step. Missing three data samples would result in a step change in position of over 0.5m. A similar issue arises if new driving commands are not issued at regular 5ms intervals, allowing the vehicle to cover significant distances with potentially incorrect trajectories. Thus it is crucial that the system maintains a consistent control signal.

1) Implementation: The High Level Controller is implemented on a 1.8GHz PC/104 computer running MathWork's xPC real-time target. The system executes a single thread with hard real-time guarantees on its execution time. At each time step, this thread processes incoming data, analyzes the path information, and outputs new driving commands. These algorithms are discussed in Section IV.

2) GPS Data Estimation: The GPS outputs data as UDP/IP packets at 200Hz, which was the limiting speed deciding the overall control structure. The GPS and High Level Controller run at this same update rate but are unsynchronized; further, the GPS does not output data at regular 5ms intervals. As a result, the High Level Controller often executes a new time step with no new data since the previous time step, or it may receive a second set of data in the same time step. To account for this inconsistency, the High Level Controller estimates critical information such as vehicle position and velocity from the last received GPS values. Given how stale a given data packet is, the High Level Controller can integrate values such as vehicle position and velocity forward in time based on acceleration to estimate the position and velocity of the vehicle at the time instant the controller executes.

Integrating these values in time requires knowing how old a given data packet is, the amount of time from packet creation by the GPS until packet use by the High Level Controller. This is determined by using UTC time stamps embedded in each data packet in conjunction with with a pulse per second (PPS) output on the GPS. The PPS is synchronized with whole number UTC seconds, which together with the embedded time stamps allows the xPC to estimate UTC time and track packet age. The resulting time difference between packet creation and use, T_{diff}, is used in a simple forward euler integration to update the most recent data to an approximation of that data at the time of the controller's execution:

$$\hat{x} = x_{in} + \dot{x}_{in} T_{diff}, \tag{1}$$

where x represents a vehicle state such as position or velocity. If no new data is received from one time step to the next, then the previous time step's data is used a second time with a different integration time. In addition to thus handling missed data on a given time step, this approximation also helps smooth out discontinuities due to inconsistent packet transmission from the GPS.

C. Safety Features

High speed driving at the limits of a vehicle's handling capability requires numerous safety features. First, the system incorporates three potential ways of stopping the vehicle in an emergency situation. Second, the Monitoring System watches the other systems and flags any potential errors, safeguarding against potential system error. And finally, the Monitoring System, which is responsible for most of the

safety features on this vehicle, itself is designed to minimize the chance of failure.

1) Emergency Stops: Three different kinds of emergency stops are available, as outlined in Table 3, which allow the vehicle to be stopped in a variety of conditions.

Softstop: The most commonly used, a softstop is executed by the High Level Controller, which steers the vehicle to remain on the path while using the remaining road friction to brake the car. The result is a smooth, controlled stop along the driving path. Softstops can be initiated from several sources. The High Level Controller itself can issue a softstop if it detects an issue with incoming data, such as consecutive unreceived data packets. If the Monitoring System detects an error, it sends a CAN message to the High Level Controller to begin a softstop. A person outside the vehicle with a Torc Safe Stop transmitter can also press a button to signal a softstop; this signal is received by the Safe Stop receiver and relayed to the Monitoring System, which in turns sends the command to the High Level Controller.

Monitor system stop: While the softstop is the most controlled stop, it depends on the High Level Controller. Should the High Level Controller go down, a monitor stop is executed by the Monitoring System. A monitor stop uses only lanekeeping feedback control to steer to the path while applying a constant braking force. The result is a simple, albeit less well executed, stop.

Hardstop: The third emergency stop, a hard stop, bypasses all of the above systems. A hard stop is initiated by a person with a Torc Safe Stop transmitter (the same device capable of issuing a softstop), and the Safe Stop receiver in the vehicle flags a low level microcontroller to shut down the engine and apply full brakes. While extreme, this method provides a guaranteed method of stopping the car in the event of a catastrophic system-wide failure.

2) Checking Subsystems: The Monitoring System's primary function is to check incoming data from the other systems for possible error conditions. If the Monitoring System detects an error condition in a system other than the High Level Controller, it issues a softstop command to the High Level Controller. As data is acquired from various systems, the information is compared against a user defined list of possible error conditions related to that data. Error conditions include error statuses in a status messages or information indicating undesirable conditions or logical errors. An undesirable condition, for example, may be if the GPS's RMS error exceeds a certain threshold. If it grows too large, we would rather stop the vehicle than continue driving. A logical error would include negative speeds or heading angles outside of 0 and 360 degrees.

The error list is implemented as a separate XML file stored in a directory on the Monitoring System. A separate, easily modifiable file allows easy modification of error conditions without modifying the Monitoring System software. This is useful in situations where the desired error conditions consistently change. For example, when testing at a new location, the threshold above which the RMS error causes a softstop may initially be higher, and as repeaters and base stations are setup for maximum coverage, the RMS error threshold can be decreased to a more demanding value.

If the Monitoring System detects that the High Level Controller stops functioning, instead of sending a softstop command, it takes over and issues driving commands to the drive-by-wire system for a monitor system stop. By using a relatively simple stopping algorithm implemented in another programming language on separate hardware, we increased the robustness of this system regardless of changes to the High Level Controller.

3) The Monitoring System: The Monitoring System, which is critical to the overall safety of the vehicle in autonomous operation, has been implemented in a separate programming language and with a software architecture designed to maximize robustness. First, the Monitoring System has been implemented in Java Real-Time on an Open Solaris operating system to reduce the likelihood of incorporating similar code or implementation errors as the High Level Controller. Second, the system is structured around software modules to ensure robust and real-time execution. Modules are priority based real-time threads providing monitor and control interfaces to a master module as well as message and data logging to a file I/O thread. Each module has a specific function, such as reading in UDP packets from the GPS or logging data to file.

The master module provides much of the Monitoring System's robustness. The master module starts, monitors, and stops all of the other modules. This module has minimal likelihood of runtime failure, with code that could cause exceptions, nondeterministic delays, or other issues moved to other modules. For example, all of the hardware initialization, which can be error-prone, occurs in the initializations of the individual modules. If one of these initializations hangs or fails, the master module has the error checking capability to recognize the issue, issue a softstop to the High Level Controller, and attempt to restart the failed module.

III. PATH DESIGN PROCESS

For driving autonomously, the vehicle uses a pre-computed optimized path which it follows based on accurate differential GPS positions. This section describes the design process for generating this path.

Beginning with the road edges of a track, creating a racing line can be divided into two parts. The first is to define the straights along the track. The second involves linking the straights with the three curve structure. The design process discussed assumes that straights along the track have already been found. Figure 4 is a example of what the straights would look like for a simple turn; however, before the straights are connected by the chosen curve structure, it is important to know how the curves are defined.

A. THREE PHASE CORNER STRUCTURE

Each turn is designed using two consecutive straights. The turn structure is a sequence of four curves: a straight, an entry clothoid, a constant radius arc, and an exit clothoid as can be seen on Figure 5. A straight also follows the exit clothoid

Stop Type	What	Executed By	Source
Softstop	Controlled stop along path	High Level Controller	High Level Controller, Monitoring System, person
Monitor stop	Semi-controlled stop along path	Monitoring System	High Level Controller crashes
Hardstop	Engine shutdown, full brakes	Microcontroller / Relay	person

Fig. 3. Available emergency stops

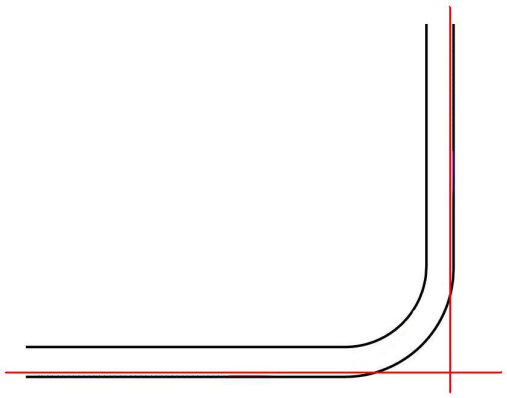

Fig. 4. *Straights defined on a simple turn*

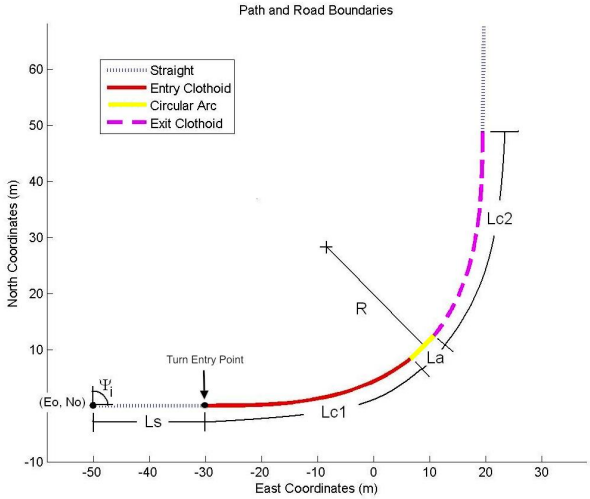

Fig. 5. *Parameters that control a turn*

which is considered part of the next curve sequence. When a turn is uniquely defined, the east and north position as a function of distance along the track is described by the following sets of equations:

Straight:

$$\begin{aligned} E_i(s) &= E_0 + s\cos(\psi_i) \\ N_i(s) &= N_0 + s\sin(\psi_i) \\ \psi_i(s) &= \psi_0, \end{aligned} \tag{2}$$

E_0, N_0, and ψ_0 is the initial position and heading of the four curve sequence. s is the distance along the track from (E_0, N_0). The end of the straight and beginning of the entry clothoid is found when $s = L_1 \equiv L_s$ where L_s is the length of the straight.

Entry Clothoid:

$$\begin{aligned} E_i(s) &= E_1 + \int_{L_1}^{s} cos(\psi_i)d\psi_i \\ N_i(s) &= N_1 + \int_{L_1}^{s} sin(\psi_i)d\psi_i \\ \psi_i(s) &= \psi_1 + \frac{(s-L_1)^2}{2RL_{c1}} \end{aligned} \tag{3}$$

E_1, N_1, and ψ_1 is the initial position and heading of the entry clothoid. L_{c1} is the length of the entry clothoid and R is the radius of curvature of the constant radius arc. The end of the clothoid and beginning of the constant radius arc is found when $s = L_2 \equiv L_{c1} + L_s$.

Constant Radius Arc:

$$\begin{aligned} E_i(s) &= E_2 + \int_{L_2}^{s} cos(\psi_i)d\psi_i \\ N_i(s) &= N_2 + \int_{L_2}^{s} sin(\psi_i)d\psi_i \\ \psi_i(s) &= \psi_2 + \frac{s-L_2}{R} \end{aligned} \tag{4}$$

E_2, N_2, and ψ_2 is the initial position and heading of the constant radius arc. L_a is the length of the constant radius arc.The end of the arc and beginning of the exit clothoid is found when $s = L_3 \equiv L_a + L_{c1} + L_s$.

Exit Clothoid:

$$\begin{aligned} E_i(s) &= E_3 + \int_{L_3}^{s} cos(\psi_i)d\psi_i \\ N_i(s) &= N_3 + \int_{L_3}^{s} sin(\psi_i)d\psi_i \\ \psi_i(s) &= \psi_3 + \frac{L_{c2}}{2*R} - \frac{(L_4-s)^2}{2*R*L_{c2}} \end{aligned} \tag{5}$$

E_3, N_3, and ψ_3 is the initial position and heading of the constant radius arc. L_{c2} is the length of the exit clothoid. The end of the exit clothoid is found when $s = L_4 \equiv L_{c2} + L_a + L_{c1} + L_s$. Note: The east and north position of (3) and (5) can be expressed as a Fresnel powers series.

B. SOLVING FOR THE PARAMETERS

Each ψ_i equation is used for simplification. Not counting the explicit ψ_i equations, there are eight equations (two from each curve) and eight parameters that uniquely constrain a given turn. E_0, N_0, ψ_0, L_s, L_{c1}, L_a, L_{c2}, and R were chosen as the parameters that will uniquely define a given turn. Each parameter defines the characteristics of how a turn is

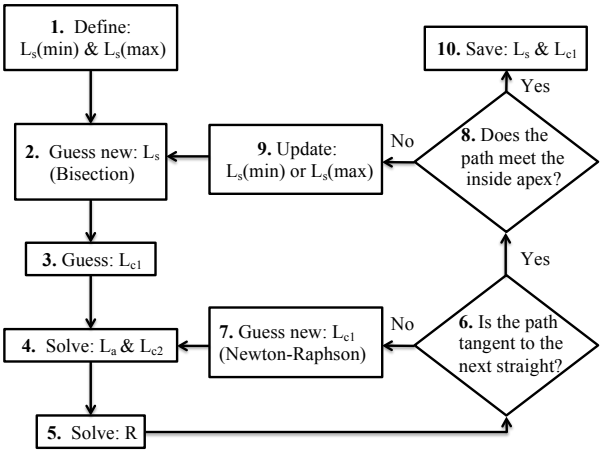

Fig. 6. *Flowchart for solving dependent parameters*

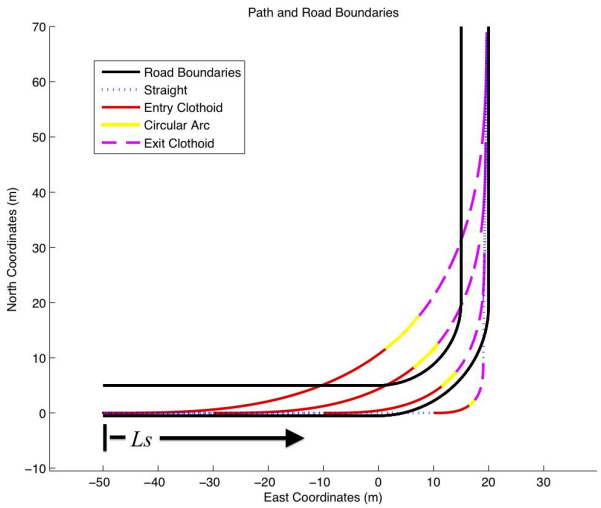

Fig. 7. *Varying L_s along a straight*

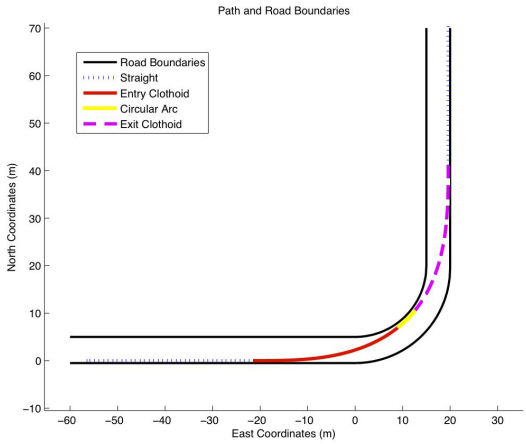

Fig. 8. *Racing line of a simple turn*

shaped as can be seen on Figure 5 and can be organized into three groups: initial conditions, independent parameters, and dependent parameters.

1) INITIAL CONDITIONS: E_0, N_0, and ψ_0 are considered the initial conditions of the turn and correspond to the beginning of the racing line or the end of a previous turn. E_0, N_0, and ψ_0, are readily found after the straights are defined along a track.

2) INDEPENDENT PARAMETERS: L_a and L_{c2} can be considered independent parameters. By relating their relative lengths to L_{c1} the shape of a turn can be varied arbitrarily, e.g.

$$\begin{aligned} L_a &= \frac{L_{c1}}{10} \\ L_{c2} &= L_{c1} \ . \end{aligned} \tag{6}$$

How these parameters are related can have a significant effect on the strategy and performance of the racing line. Making L_{c1} larger or smaller than L_{c2} would correspond to late or early apex turns. The length of L_a controls how long a racecar travels at maximum cornering. This parameter can be used to switch between two or three phase turns. If L_a is set to zero then the turn has two phases and reaches maximum cornering for only an instant.

3) DEPENDENT PARAMETERS: The workflow diagram on Figure 6 illustrates how the dependent parameters are solved in order to meet two requirements. The first is to ensure that the end of the turn or exit clothoid is tangent to the following straight in order to properly connect each straight with the next one in the sequence. The second is for the apex of the racing line to meet the inside boundary of the turn in order to minimize the curvature of the turn while remaining within the road boundaries.

1. Define L_s(min) and L_s(max) - L_s is the point along the straight where the turn begins. It can vary along a straight as demonstrated in Figure 7. From the figure, one can conclude that L_s is bounded by the beginning of a straight (E_0, N_0) and the point where the two straights intersect. Because the range of L_s is known, bisection is used to find the value of L_s that will allow the apex of the turn to meet the inside boundary.

2. Bisection: guess new L_s - This block begins the outer loop that is used to solve for L_s. Each new guess of L_s is half way between the current L_s(min) and L_s(max).

3. Guess L_{c1} - The initial guess of L_{c1} is arbitrary as long as it is a real, positive number. The guess initializes the inner loop that solves for L_{c1} given a specific L_s.

4. Solve for L_a and L_{c2} - These lengths are readily found using (6).

5. Solve for R - The radius of the circular arc, R, is found from the difference in heading between the two straights:

$$\begin{aligned} \Delta\psi &= \psi_{i+1} - \psi_i \\ &= \frac{L_{c1}}{2R} + \frac{L_a}{R} + \frac{L_{c2}}{2R} \\ \Rightarrow \quad R &= \frac{1}{2\Delta\psi}(L_{c1} + 2L_a + L_{c2}) \end{aligned} \tag{7}$$

6. Is the path tangent to the next straight? - At the end of the exit clothoid, the racing line must be tangent to the straight in the next sequence. The equation for the next

straight can be described by:

$$N_{i+1} = E_{i+1} * tan(\psi_{i+1}) + b_{i+1} \quad (8)$$

where b_{i+1} is the location of the intercept on the N axis. The end of the exit clothoid has a final position and heading that can be used to define a line as well. If both intercepts are equal, then the path is tangent to the next straight.

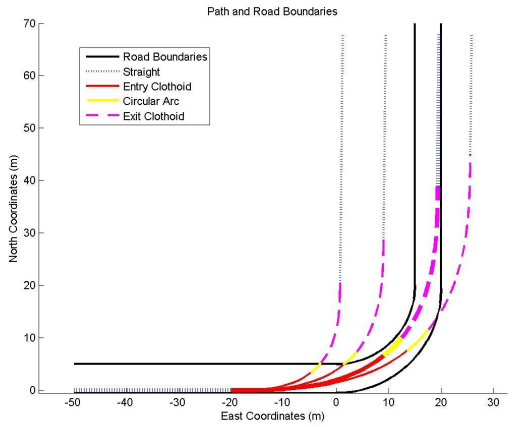

Fig. 9. *Varying the curve lengths to ensure tangency*

7. Guess new L_{c1} - If the racing line is not tangent to the next straight, Newton-Raphson is used to ensure that the N intercept of the racing line matches b_{i+1} by adjusting the length of L_{c1} and subsequently L_a, and L_{c2}. Figure 9 is an example of varying the curve lengths to fit the racing line to the desired straight.

8. Does the path meet the inside apex? - Figure 8 is an example of the path meeting the inside of the road boundary at the apex. This condition can be determined by calculating the distance from the path to the inside road boundary.

9. Update L_s(min) or L_s(max) - If the turn begins too early the lower bound $L_{s,min}$ is updated otherwise $L_{s,max}$ is updated. Figure 7 is an example of how the solution of L_s would converge.

In summary, a combination of bisection and Newton-Raphson is used to solve for the constraint parameters of each turn along the track. This process is used to both minimize the curvature of each turn and ensure that the path is tangent to the following straight.

IV. HIGH LEVEL CONTROLLER: CONTROLLING AT THE LIMITS

From the path described in Section III, the high level controller calculates the steering, throttle and brake inputs in real-time to fully utilize the tire forces while tracking the path.

A "g-g" diagram (see Fig. 10) is used to quantify the friction limits [4]. To simplify the concept for illustrative purposes, a friction circle (or a traction circle) can be used to define the limits on a "g-g" diagram (see Fig. 10) when ignoring the effect of weight transfer and when the tires have isotropic property.

Alternatively, the high level controller in this paper uses a boundary that takes the effect of longitudinal weight transfer into account. The friction coefficient is estimated from a ramp-steer maneuver. Thus, the objective of the controller is to follow the desired path in Section III while operating at the friction limits defined by a "g-g" diagram as demonstrated in the experimental data in Fig. 11.

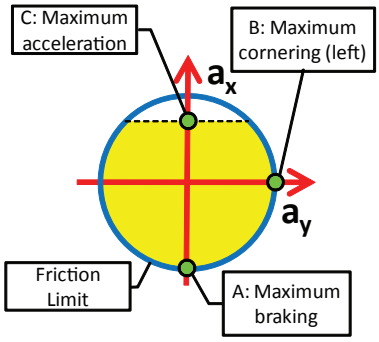

Fig. 10. Using a "g-g" diagram to describe vehicle limits

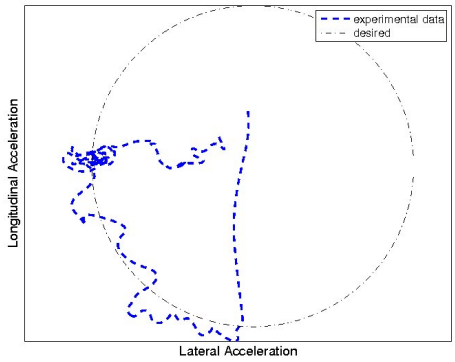

Fig. 11. "g-g" diagram measured at Pikes Peak dirt section

To track the path at the friction limits, the high level controller is separated into feedforward and feedback modules, with each module consisting of lateral and longitudinal controllers (see Fig. 12). The feedforward controllers calculate the steering and longitudinal (throttle and brake) inputs that drive the vehicle along the path and make the vehicle's acceleration trace the friction limit on a "g-g" diagram. While driving, the feedback controllers make adjustments based on vehicle responses and tracking errors. The map matching block shown in Fig. 12 converts GPS measurements into vehicle states referenced to the desired path. The detailed explanation of this high level controller can be found from [1], [2]

A. Feedforward Controller

Similar to drivers planning their inputs before going into a corner, the feedforward controllers calculate the amount of steering, throttle and brake in advance.

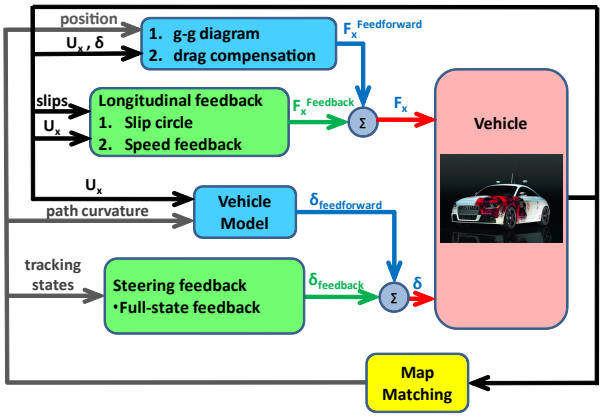

Fig. 12. Overall controller block diagram

1) Feedforward Longitudinal Controller: The feedforward longitudinal controller calculates the amount of throttle and brake while driving along the path. Using the path curvature and the friction limit on a "*g-g*" diagram that takes longitudinal weight transfer into account, real-time speed and acceleration profiles along the path are calculated [1]. The feedforward longitudinal force is then computed from the desired longitudinal acceleration profile. In addition, drag compensation is added into the feedforward longitudinal command to minimize the effects from aerodynamic forces, rolling resistance, etc.

2) Feedforward Lateral Controller: The feedforward lateral controller predicts the amount of steering input δ from the vehicle speed U_x, path curvature and vehicle parameters. The algorithm uses a bicycle model with nonlinear tires to calculate the steering input [2].

B. Feedback Controller

When testing the algorithm in an actual environment, feedback controllers are used to minimize the effects of modeling error and disturbances in the system. The feedback controllers also improve tracking ability as well as provide system's stability.

1) Feedback Longitudinal Controller: The feedback longitudinal controller regulates tire slip through modulating longitudinal input [1]. It uses a slip circle to detect tire slip. A slip circle is a plot of a normalized combined lateral and longitudinal slip. Lateral slip is calculated from GPS/INS measurement and front steering angle, while longitudinal slip is calculated from GPS/INS measurement and measured wheel speed. Any combined slip on the unit circle produces peak tire forces. Thus, the goal of this longitudinal slip circle feedback controller is to modulate the longitudinal input to bring the combined slip back into the unit circle. For instance, if the slip circle detects excessive braking, it will reduce the amount of brake. To simplify the slip circle algorithm, the feedback longitudinal controller monitors front and rear slip circles, rather than individual tire slip.

In addition to the longitudinal slip circle feedback, the speed feedback is used to ensure that the vehicle speed tracks the desired speed calculated in Section IV-A.

2) Feedback Lateral Controller: The feedback lateral (steering) controller provides path tracking and stability for the system by regulating lateral and heading errors. A set of fixed gains full-state feedback is used, and the system is proved to be Lyapunov stable even when the rear tires are highly saturated [2].

V. Conclusions and Future Work

In this paper we presented a novel system enabling a vehicle to autonomously operate at the limits of friction, which will have useful implications in future collision avoidance and accident prevention systems. Compared to other research in this area, our system is designed for high speed real-time control at 200 Hz, tracking a precomputed clothoid based optimized trajectory at the limits of the vehicle's capability. The vehicle was tested at and successfully completed the Pikes Peak Hill Climb course, which incorporates a combination of paved and unpaved road surfaces, significant bank and grade, and narrow operating areas.

Future work includes runtime modification of the desired path. Paths could be modified to avoid an obstacle, for example, or a new path could be calculated if the vehicle exceeds its limits. Road friction could also be estimated real-time, which would improve the accuracy of the controller.

VI. Acknowledgments

This research is supported by the Volkswagen Automotive Innovation Lab (VAIL) and Electronics Research Laboratory of Volkswagen of America (ERL) including Sven Beiker, Robin Simpson, Robert MacLellan, Pablo Marx and Jens Kotte. The authors would also like to thank Dynamic Design Lab colleagues David Hoffert and Kirstin Talvala for their support of the project, the City of Mountain View, CA for their assistance with testing and Omnistar, Inc. for providing their DGPS service. Furthermore thanks to Mike Duigou and Ivan Tarasov from Oracle for their support on the Java system and vehicle testing.

Joseph Funke is supported by a Graduate Research Fellowship from the National Science Foundation.

Krisada Kritayakirana is supported by a Fulbright Science and Technology Fellowship.

References

[1] K. Kritayakirana and J. C. Gerdes, "Autonomous Vehicle Control at the Limits of Handling", *International Journal of Vehicle Autonomous Systems (IJVAS): Special Issue for 10th Anniversary*, in press.

[2] K. Kritayakirana and J. C. Gerdes, "Using the Center of Percussion to Design a Steering Controller for an Autonomous Racecar", *Vehicle System Dynamics*, in press.

[3] M. Montemerlo et. al., "Junior: The Stanford Entry in the Urban Challenge", *Journal of Field Robotics*, Vol. 25, Issue 9, 2008, pp 569-597

[4] R. S. Rice, "Measuring Car-Driver Interaction with the g-g diagram", *Society of Automotive Engineers*, Warrendale, PA, 1973, pp. 1-19.

[5] G. Stanek et. al., "Junior 3: A Test Platform for Advanced Driver Assistance Systems", *Proceedings IEEE Intelligent Vehicles Symposium*, San Diego, CA, June 2010, pp 143-149

[6] U.S. Department of Transportation National Highway Traffic Safety Administration, *Traffic Safety Facts 2009*, A Compilation of Motor Vehicle Crash Data from the Fatality Analysis Reporting System and the General Estimates System, Number DOT HS 811 402. 2011.