# Trajectory Tracking Control of an Autonomous Ground Vehicle

**Thesis** · January 2008

DOI: 10.13140/RG.2.2.34957.03041



**1 author:**

Kristian Soltesz
Lund University
**45** PUBLICATIONS **301** CITATIONS

ISSN 0280-5316
ISRN LUTFD2/TFRT--5810--SE

# Trajectory Tracking Control of an Autonomous Ground Vehicle

Kristian Soltesz

Department of Automatic Control
Lund University
February 2008

**Lund University**
**Department of Automatic Control**
**Box 118**
**SE-221 00 Lund Sweden**

*Document name*
MASTER THESIS

*Date of issue*
February 2008

*Document Number*
ISRN LUTFD2/TFRT--5810--SE

*Author(s)*
Kristian Soltesz

*Supervisor*
Richard M. Murray at Caltech, USA
Tore Hägglund at Automatic Contron. Lund (Examiner)

*Sponsoring organization*

*Title and subtitle*
Trajectory Tracking Control of an Autonomous Ground Vehicle (Banföljning vid körning med autonoma markfordon)

*Abstract*
This thesis proposes a solution to the problem of making an autonomous nonholonomic ground vehicle track a special trajectory while following a reference velocity profile. The proposed strategies have been analyzed, simulated and eventually implemented and verified in Alice, Team Caltech's contribution to the 2007 DARPA Urban Challenge competition for autonomous vehicles.
The system architecture of Alice is reviewed. A kinematic vehicle model is derived. Lateral and longitudinal controllers are proposed and analyzed, with emphasis on the nonlinear state feedback lateral controller. Relevant implementation aspects and contingency management is discussed. Finally, results from simulation and field tests are presented and discussed.

*Keywords*

*Classification system and/or index terms (if any)*

*Supplementary bibliographical information*

*ISSN and key title*
0280-5316

*ISBN*

*Language*
English

*Number of pages*
57

*Recipient's notes*

*Security classification*

# Contents

# List of Figures

# Nomenclature

**Abbreviations**

| | |
|---|---|
| AGV | Autonomous Ground Vehicle. |
| CSS | Canonical Software Structure. Framework defining software module- and interface structure. |
| DGC | DARPA Grand Challenge. Competition for autonomous ground vehicles held in a desert environment. |
| dGPS | Differential GPS. An augmentation to GPS, consisting of a base station which sends corrections to the GPS signal. |
| DMI | Distance Measurement Indicator. Wheel mounted rotation sensor. |
| DUC | DARPA Urban Challenge. A competition for autonomous ground vehicles, held in an urban environment. |
| E-stop | Emergency stop signal issued by DARPA. The E-stop was the only means of communication with the competing vehicles during competition runs. |
| GPS | Global Positioning System. A satellite based localization system. |
| IMU | Inertial Measurement Unit. A system of gyroscopes and accelerometers used for vehicle state estimation. |
| IPT | Integrated Project Team. Sub-team of coordinators within Team Caltech. |
| JPL | Jet Propulsion Laboratory of the National AeroSpace Agency (NASA). |
| KVM | Keyboard Video Mouse. A switch to which computers and I/O are connected. The user can choose which connected computer to control, without having to re-route keyboard-, monitor- and mouse cables. |
| LADAR | LAser Detecton And Ranging. Sweeping sensor used for obstacle detection. |
| MDF | Mission Data File. A file defining the mission by means of an ordered list of checkpoints located within the RNDF. |
| NQE | National Qualifying Event of the DARPA Urban Challenge. |
| PCM | Powertrain Control Module. |
| PLC | Programmable Logic Controller. |
| RNDF | Route Network Definition File. A file containing a vector map of the route network. |
| SVN | Subversion. A version management software. |
| UTM | Universal Transverse Mercator. A grid based coordinate system for specifying locations at the surface of the earth. |
| wiki | A collaborative website. Its content can be edited by anyone who has access to it. |
| YaM | Yet another Make. A combined version manager and make utility. |

**Symbols**

| | | |
|---|---|---|
| $\delta$ | Angle between trajectory tangent at $R$ and steering wheel heading (of the real vehicle). | rad |
| $\dot{\phi}_{\max}$ | Maximal allowed steer rate being a function of vehicle speed $v$. | rad $s^{-1}$ |
| $\hat{\tau}_a$ | Acceleration actuation delay estimate. | s |
| $\omega_c$ | Cutoff 'frequency' of the linearized system. | rad $m^{-1}$ |
| $\phi$ | Front wheel angle. | rad |
| $\phi_m$ | Phase margin of the linearized system. | rad |
| $\phi_v$ | Front wheel angle of virtual vehicle. | rad |
| $\phi_{\max}$ | Angle between front wheel- and vehicle heading corresponding to $u_\phi = -1$. | rad |
| $\phi_{\min}$ | Angle between front wheel- and vehicle heading corresponding to $u_\phi = -1$. | rad |
| $\tau_a$ | Acceleration actuation delay. | s |
| $\tau_m$ | Delay distance margin of the linearized system. | m |
| $\tau_\phi$ | Steering actuation delay. | s |
| $\theta$ | Angle coordinate of vehicle point $P$. | rad |
| $\theta_r$ | Angle coordinate of reference trajectory tangent at $R$. | rad |
| $\theta_r$ | Trajectory point reference heading. | rad |
| $a$ | Acceleration of the vehicle in the direction of its heading | $ms^{-2}$ |
| $A_c$ | System matrix of the linearized closed loop system. | - |
| $A_o$ | System matrix of the open loop linearized system. | - |
| $a_r$ | Acceleration reference. | m $s^{-2}$ |
| $a_u$ | Desired acceleration of the vehicle in the direction of its heading. | $ms^{-2}$ |
| $B_c$ | Input matrix of the linearized closed loop system. | - |
| $B_o$ | Input matrix of the open loop linearized system. | - |
| $C$ | Output matrix of the linearized (closed loop) system. | - |
| $C_n$ | Constant or parameter, with integer index $n$. Constant names are reused throughout the text. | - |
| $d$ | Travelled distance. | m |
| $d_i$ | Distance between consecutive trajectory points. | m |
| $d_{FF,a}$ | Acceleration delay feed forward distance. | m |
| $d_{FF,a}$ | Steering delay feed forward distance. | m |
| $e$ | Easting of trajectory point. | m |

| | | |
|---|---|---|
| $e_y$ | Perpendicular error. Distance from the center of the rear axle to its projection $R$ onto the reference trajectory. | m |
| $e_\theta$ | Yaw error. Angle between the heading of the vehicle and the tangent of the reference trajectory at $R$. | rad |
| $G$ | Closed loop transfer function from $y_r$ to $y$. | m→m |
| $G_o$ | Open loop transfer function of linearized system from $e_y$ to $y$. | m→m |
| $I_v$ | Speed error integral. | - |
| $L$ | Wheel base. | m |
| $l_2$ | The only tunable parameter of the lateral controller. | m |
| $m$ | Maximal time derivative order of reference trajectory point coordinates. | - |
| $n$ | Northing of trajectory point. | m |
| $P$ | Rear axle center. | - |
| $R$ | Projection of the rear axle center $P$ onto the reference trajectory. | - |
| $r$ | Turning radius. | m |
| $r_c$ | Radius of circular reference trajectory. | m |
| $r_{\min}$ | Minimal turning radius of Alice (7.35 m). | m |
| $u_a$ | Acceleration control . | $\in [-1,1]$ |
| $u_\phi$ | Steering control signal. | $\in [-1,1]$ |
| $v$ | Vehicle speed. | $\mathrm{ms}^{-1}$ |
| $v_r$ | Reference speed. | $\mathrm{ms}^{-1}$ |
| $y$ | Perpendicular coordinate of vehicle point $P$. | m |
| $y_r$ | Perpendicular coordinate of reference trajectory point $R$. | m |

# Acknowledgments

The work resulting in this thesis was carried out at the Department of Control and Dynamical Systems, California Institute of Technology, Pasadena, USA with additional supervision from the Department of Automatic Control, Lund Institute of Technology, Lund, Sweden. It was part of *Team Caltech*'s autonomous vehicle program, aimed at the 2007 DARPA Urban Challenge.

I would like to thank my supervisor and the coordinator of Team Caltech, Professor Richard Murray, California Institute of Technology. Additionally, I would want to thank my co-supervisor Professor Tore Hägglund and Professor Anders Rantzer at Lund Institute of Technology, Lund, Sweden together with the Caltech Summer Undergraduate Research Fellowship (SURF) program for making my stay in California both possible and pleasant.

Last, but not least, I owe great thanks to all members of Team Caltech – for their help, support and suggestions throughout the project – as well as all external supporters and sponsors of the project.

# 1. Introduction

The concepts, ideas and results presented here are the outcome of Team Caltech's development for the 2007 DARPA Urban Challenge (DUC) [15]. The aim of the project was to develop an autonomous ground vehicle (AGV) for safe operation in an urban environment – in the presence of other vehicles. The final evaluation of this effort took place at the National Qualifying Event (NQE) of the DUC, see Section 8.2. A photograph of Team Caltech's vehicle, Alice, is shown in Figure 1.1.


**Figure 1.1** Photograph showing Team Caltech's DUC vehicle Alice.

The focus of this paper lies on the trajectory tracking controllers which were developed, implemented, and tested by me and Magnus Linderoth from Lund Institute of Technology. However, it is important to emphasize that the development process was a team effort. Consequently, a summary of previous work conducted at Caltech, as well as a review of current team- and system structure, are given before focus is shifted to the trajectory tracking controller module.

A motivation for the DUC is given in Section 1.1, followed by a brief history of previous races in Section 1.2. The organization of this paper is then presented in Section 1.3.

## 1.1 Motivation

The aim of the DUC was to push the development of AGV systems, capable of performing safely in an urban environment – in the presence of other vehicles.

The main motivation for Team Caltech, the Stanford Race Team and others was to develop fully autonomous vehicles, which are safer than their human maneuvered counterparts, hence increasing traffic safety. A large amount of research and development has been made during the past few years, but there will still be some time before the technologies are ready for the consumer market. A more realistic short term application are systems, which prevent a human driver from making 'illegal' maneuvers, by e.g. clearing (suddenly appearing) obstacles, stopping at stop lines, etc. Such and similar applications are discussed in [19, 22].

Another category of teams, e.g. the Oshkosh Truck Company, aim to use their findings in military applications. Their main objective has been to produce reliable AGV transportation systems, which could be used in hazardous areas.

Independent of application, autonomous driving is a complex applied robotics problem, stretching over diverse fields of engineering, including controls, signal processing, computer vision, computer science and mechanical engineering – to mention some of the major ones. To further complicate matters, the urban environment is rich in both detail and unexpected situations, calling for highly robust systems.

Apart from the technical aspects, participating in Team Caltech was a great opportunity for students to work in an environment otherwise rear in academia. The work was project oriented, highly applied and involved hard deadlines.

## 1.2 The DARPA Grand Challenge

The 2007 DUC was the third in a series of autonomous road vehicle competitions organized by DARPA. The previous two went under the name DARPA Grand Challenge (DGC) and took place in 2004 [14] and 2005 [16]. Team Caltech was formed in March 2003 in order to compete in the 2004 DGC and has also contributed with an entry for the 2005 competition.

The 2007 DUC consisted of a series of qualifying steps, leading to a final event. These steps are shown in Figure 1.2. (Although details have changed, most of the following applied also to the 2004 and 2005 races.)

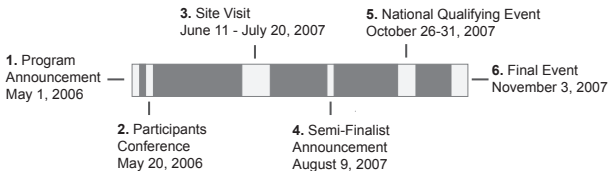



**Figure 1.2** Time line of the 2007 DUC.

Here follows a short description of each step of the time line in Figure 1.2.

1. *Program Announcement May 1, 2006*. DARPA announced the schedule for the 2007 DUC. New for the 2007 DUC was that 11 teams, including Team Caltech, were to receive funding of $ 1 M each, for developing their vehicles.
2. *Participants Conference May 20, 2006*. The Participants Conference served as a briefing session, informing the teams what to expect during the remainder of the DUC.
3. *Site Visits June 11 - July 20, 2007*. The test sites of the competing teams were visited by DARPA officials. During these visits the teams had to demonstrate safe operation and basic functionality of their vehicles.
4. *Semi-Finalists announcement August 9, 2007*. Based mainly on the outcome of the site visits, 34 teams were recognized as semi finalists. Team Caltech was one of the semi-finalist teams.
5. *National Qualifying Event October 26 - 31, 2007*. The NQE served as a series of 'unit tests', where the vehicles needed to demonstrate successful handling of advanced traffic scenarios (involving several vehicles).
6. *Final Event November 3, 2007*. Each team that qualified to the final event was provided with a USB stick holding two files from DARPA. These were the Route Network Definition File (RNDF) and the Mission Data File (MDF) [17].

The RNDF was a 'map', where lane geometry, stop lines, intersections, waypoints etc. were given in a frame fixed to absolute GPS coordinates. The MDF was an ordered list of GPS waypoints located within the area of the RNDF. It also defined speed limits for all road segments of the RNDF. The objective of the competition was to traverse the checkpoints of the MDF in correct order and minimal time, while adhering to traffic rules [18], provided by DARPA. All finalist ran on the same RNDF – simultaneously.

### Grand Challenge 2004

The 2004 DGC attracted 109 teams, from all over the USA. Out of these, 25 teams were recognized as finalists. Team Caltech made it to the final event with the modified Chevy Tahoe *Bob*, shown in Figure 1.3. Not unlike Alice, Bob utilized laser detection and ranging (LADAR) sensors and stereo vision for sensing and a combination of GPS and an Internal Measurement Unit (IMU) for vehicle state estimation. More information on Bob is given in [26].


**Figure 1.3** Photograph showing Team Caltech's 2004 DGC vehicle Bob. Courtesy of DARPA.

The final event was held on March 13, 2004, in the Nevada desert. None of the starting teams finished the course. The Red Team [4] got furthest by completing 11.8 km, which was less than 5 % of the entire course. Team Caltech's Bob completed 2.0 km of autonomous operation before running into a barbed wire fence and aborting.

### Grand Challenge 2005

In the 2005 DGC, 43 out of 195 teams (of which 4 were non-US) made it to the NQE. Out of these, 23 teams, including Team Caltech, qualified to the final event.

Team Caltech's contribution to the 2005 DGC was the modified Ford E-350 van *Alice* [27] shown in Figure 1.1. From a practical point of view Alice had at least two clear advantages over Bob:

- *Space.* Alice was bigger, allowing up to eight persons to sit inside and monitor or develop the system in realtime.
- *Manual mode.* Bob did not have a manual drive mode and needed to be towed to and from test sessions.

The final event of the 2005 DGC was held on October 9, 2005. The location was, again, a desert area in Nevada. The outcome was, however, very different. This time five vehicles completed the 212 km race course. Only one vehicle failed to pass the 11.8 km mark set by the furthest going 2004 entry. The fastest time was set by the Stanford Racing Team [5]. Their vehicle completed in 6 hours and 45 minutes (average speed: 31 km $h^{-1}$) and won the $ 2 M Grand Prize.

Team Caltech's Alice started out with all systems functioning, but lost its mid range LADARs approximately 4 minutes into the race. After an additional 26 minutes, Alice went under a set of power lines, losing GPS signal. When the signal was regained the GPS estimates reported large signal errors. This led to slow state convergence, which was mistaken for the state having converged. With incorrect state estimates and non-functioning mid-range sensors, Alice picked up speed and ran into a concrete barrier after 13 km, 32 minutes into the race. See link from [31] for images of the spectacular crash.

#### Grand Challenge 2007 – an Urban Challenge

What really put the 2007 DUC apart from previous DARPA challenge races, was the urban environment setting. Despite a large set of new difficulties, 63 teams qualified for site visits. Out of these, 34 teams, including Team Caltech, made it through their site visits. The NQE was passed by 11 teams. Unfortunately Alice failed a merging test and was therefore disqualified from the Final Event. See Section 8.2 for a detailed description of what went wrong.

The final Event was won by one of two favorites, the Tartan Racing Team (Carnegie Mellon University) [7]. The other favorite, Stanford Racing Team [5], took the second place, followed by the Virginia Tech Team [9].

## 1.3 Thesis Outline

The material found in this thesis is the result of a highly practical development process. Although trajectory tracking control of nonholonomic vehicles is a theoretical subject, most project time and effort was spent on practical matters, e.g. communicating within the team, implementing, testing and debugging. When writing this thesis, I have tried to capture the nature of the project, rather than focusing solely on the control problem. Throughout the text, subjects appear in the same chronological order as they were introduced in the project.

In Chapter 2 the structure and routines of Team Caltech are presented. A review of the team structure is followed by an introduction to key development utilities. At the end of the chapter, the development schedule for the 2007 DUC is given. Chapter 3, reviews the hardware of Alice, as well as the main features of its sensing and navigation systems. The vehicle control problem, being the topic of the remainder of the thesis, is outlined in Chapter 4. In Chapter 5 the dynamics of the vehicle are discussed, followed by the introduction of the utilized control strategies. The longitudinal (speed) controller implements a conventional strategy and is therefore given only brief attention. The lateral (steering) controller is, however, based on a novel idea and further analyzed in Chapter 6. Chapter 7 deals with practical implementation issues. Field test results, showing the performance of the lateral controller, are presented in Chapter 8. Finally, Chapters 9,10 are dedicated to conclusions and suggested future work, respectively.

# 2. Team Organization

One of the main challenges in the project was that of team coordination. Having individual developers wait for each other due to dependency issues, was avoided as far as possible due to the given tight time frame. In the same fashion, good inter-team communication was necessary to maintain coherence in the system development. This required all developers to both comment code rigorously and stay up to date with code changes. It was all facilitated by a clear team structure with well-established communication channels.

An overview of the team structure is given in Section 2.1. Section 2.2 reviews the key utilities for team coordination are reviewed. All parts of the project were more or less influenced by its tight development schedule, reviewed in Section 2.3.

## 2.1 Team Structure

Team Caltech consisted mainly of students from Caltech and visiting students from other universities. Team members also included Caltech faculty as well as NASA Jet Propulsion Laboratory (JPL) and Northrop Grumman Corp. staff. Over the years a lot of people have joined, worked in and left Team Caltech. Consequently, a significant part of the challenge was to maintain knowledge and experience within the team, although team members were frequently being exchanged.

The team actively working on preparing Alice for the 2007 DUC consisted of 77 persons, listed in Appendix B. The team was divided into several sub-teams, each with a well-defined purpose. These sub-teams are the subject of the next few paragraphs. The two biggest – in terms of persons, code as well as functionality – are given extra attention.

### Sensing Sub-Team

Alice made extensive use of sensors [13] including LADAR, radar and stereo cameras, in order to build a map of its environment. See Section 3.1 for further details. The task of the sensing team was to develop and maintain software which took information from the sensors, and fused it into the map. A big part of this work lay in segmenting data points into objects and classifying these objects as cars, static obstacles, lane lines, etc.

### Navigation Sub-Team

The navigation sub-team developed the planning software of Alice. The purpose of planning was to frequently quarry information from the RNDF, MDF and the map, as well as vehicle state information supplied by a state estimator in order to generate feasible trajectories. The trajectories were then sent to the trajectory tracking controller module, being the main subject of this thesis.

### Other Sub-Teams

Apart from sensing and navigation, there were four sub-teams:

- *Integrated Project Team (IPT).* This sub-team consisted of all the sub-team coordinators, as well as other 'key' persons. Its responsibility was to manage the project at an overview level.

- *Systems Sub-team*. All the sub-teams wrote their own programs, executed in separate processes. The systems sub-team maintained the interfaces used to enable inter-process communications. More information is given in Section 7.1.
- *Vehicle Sub-team*. The vehicle sub-team was responsible for all mechanical work and wiring on the vehicle.
- *Sysadmin Sub-team*. The sysadmin sub-team managed the networked workstations, as well as the 'race' computers in Alice.

## 2.2 Development Utilities

The following paragraphs describe six tools which were extensively used throughout the development process. They are mentioned here, because of their contributions in terms of defining standards and significantly increasing development efficiency.

### Team Wiki

Throughout the project the team wiki [8] hosted descriptions of all software modules, meeting notes, test schedules, etc. The wiki structure – version managed and editable by all project members – proved to be successful, mainly because of the rapid development process.

### Doxygen

Doxygen [2] is a documentation system for a number of languages, including C, C++ and Java. It can generate both an online documentation browser (in HTML) and offline documentation (in LaTeX). The documentation is extracted directly from the source code on which doxygen is run. (Specific tokens are used to trigger the doxygen parser.)

The project utilized the HTML documentation, which was maintained for all software modules. Apart from comments, the auto-generated documentation showed code structure, relation between objects and inter-file dependencies in an intuitive and graphically appealing format.

### Bugzilla

Bugzilla [1] is a data base with a HTML front end, used to keep track of bugs. As identified bugs were inserted into the Bugzilla database, they could be assigned to a sub-team, or a specific developer. Managed bugs were associated with a severity, as well as possible dependency relations with other bugs.

### Subversion

Subversion (SVN) [6] is a widely used version management software. File trees (or parts thereof) could be added to an SVN repository, kept on a backed up server. Each developer worked on a local copy of the code, a sandbox, and could at any given time commit files to the repository. SVN kept track of commits and gave the developer the option to revert files to any previously committed version. There was also a web front end, which enabled developers to examine the version history of files without having to check them out from the repository.

### YaM

Whenever a developer came to a point where an owned module was stable, release of the module was made, making it officially available to other developers on the team. Releases were handled by YaM [21, 10]; a combined version manager and make utility developed at JPL. It's functionality was similar to that of SVN. However, it operated on a module level, as opposed to a file level. It also implemented a custom make utility, used to build and link YaM-managed modules.

When a release of a software module was made, YaM checked that no changes had been made to the particular module after it's files were latest committed to the SVN repository. It also automatically generated a message with change-log entries and posted it on the 'implement' mailing list, informing all developers that a new module release had been made. The actual release consisted of a source part and a link part. The source part was simply the module source code, whereas the link part consisted of binaries, libraries, headers and configuration files. Its purpose was to spare developers the time it took to check out and compile a module each time there was a new release of it.

### Mailing Lists

The implement mailing list served as the main communication channel between developers. Whenever a YaM release was made, the release notes were automatically posted on this list. Notifications of meetings, schedule changes etc. were also posted here. In addition to the implement mailing list, each sub-team had its own list, for matters not affecting the entire team.

## 2.3 Development Schedule

Based on experience from the previous DGC races, Team Caltech began implementing new software modules for the 2007 DUC in June 2007. A mostly stable hardware setup was already in place. (It was, however, augmented with new computer blades and sensors. In addition to this, various electro-mechanic parts were replaced.) The vast majority of development time was therefore spent implementing and testing new functionality. This section gives an overview of the project time line and development methodology, utilized by the team in order to maximize development efficiency.

### Project Time Line

The nominal project time line for the 2007 DUC preparations was divided into five spirals. The goals for the individual spirals are reviewed in Appendix A.

Keeping up with the nominal time line turned out to be hard. There were a lot of minor issues, which together ended up taking longer time than expected to resolve. A nominal version of almost all modules, including the controller described herein, was in place by the end of spiral 1. However, Spiral 2 took longer time than planned for, making spirals 3 and 4 more stressful than originally intended. Nevertheless, by the end of spiral 4, Alice was capable of handling all the situations postulated by DARPA – but some of the functionality had not been thoroughly tuned and tested in the field prior to the NQE.

**Weekly Meetings**

Each week the entire project group met. These meetings were coordinated by the team leader, Prof. Richard Murray, and their structure were as follows:

- Review of what had happened since the last project meeting.
- Overview of the current project status with respect to the nominal time line.
- Determination of a plan for the coming week.
- Outline of a plan for the next few weeks.

In connection to the weekly project meeting, there was a weekly sub-team meeting. The structure was similar to the project meeting, but the scope was narrower, with focus on practical implementation and testing issues.

**Code Development**

Although some code development was made in the vehicle, most development and debugging took place in a lab reserved for the project. A utility, which was extensively used, was the system simulator. It enabled real time testing of all software (with minor exceptions), without the need to take Alice out of the workshop.

**Code Reviews**

One software module was reviewed by its owner(s) each week. These reviews were very helpful in giving developers insight into modules which they had a dependency relation to, but were not personally developing.

**Field Tests**

Once new code had been verified in simulation, it was field tested in Alice. Although the simulator revealed many bugs, it was not perfect and a lot of bugs were first observed in the field. Consequently, a big effort was made to maximize test time. Team Caltech had access to three test sites:

- *St. Luke Hospital Parking Area.* This site had streets with painted lane- and stop lines. There were also some buildings, bushes and trees, putting the sensing sub-system to a test. Unfortunately, the site was rather small.
- *Santa Anita Race Track parking Area.* Being the parking lot of a big race track, the site was a huge paved area, with nothing on it except for some painted lane lines. It was used mainly to test new code at higher speeds or to unit test specific parts of the sensing sub-system.
- *El Toro Former Military Base.* The El Toro base was the site most extensively used towards the end of the project. It was a large area with real roads, parking lots and several structures.

As indicated, the choice of test site depended on the particular feature(s) to be tested. While the parking lots were suitable for controller tuning, basic traffic scenarios etc., the El Toro site provided a preview of what was expected for the NQE.

# 3. System Structure

This chapter gives an overview of the system architecture of Alice. The hardware is reviewed in Section 3.1. The structure and functionality of the sensing and navigation sub-systems are then explained in Sections 3.2,3.3, respectively.

## 3.1 Hardware

A complete set of sensing-, computation- and actuation hardware was inherited from the 2005 DGC. However, some of it needed to be replaced since it would not meet the new and higher demands of the urban environment. There was also a need of augmenting the sensor array, in order to obtain both better coverage and higher resolution. The following description is aimed at the final 2007 DUC configuration.

### Vehicle State Estimation

The system for vehicle state estimation was the Applanix POS LV 420 by Trimble [12], providing estimates of vehicle position, orientation, speed and acceleration. The system consisted of a Global Positioning System (GPS) receiver for coarse position estimation. Differential GPS (dGPS) was then used to increase both accuracy and precision of the raw GPS estimate. The GPS and dGPS signals were filtered with the output of an IMU and a wheel-mounted distance measurement indicator (DMI) . The obtained state estimates had a root mean square position- and yaw errors of 0.300 m and 0.020°, respectively. A schematic drawing of the state estimation system is shown in Figure 3.1.

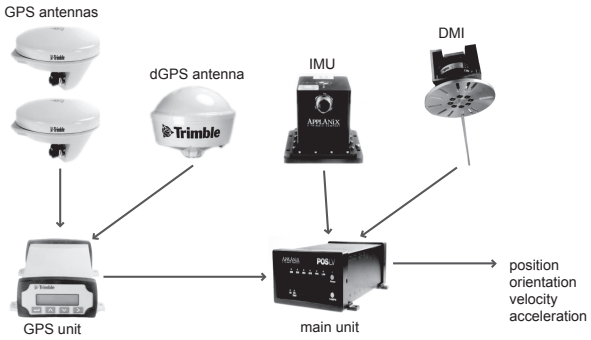



**Figure 3.1** Block diagram illustrating the Applanix POS LV 420 vehicle state estimation system.

### Sensors

An extensive array of sensors was used in order to obtain both adequate coverage and resolution. The sensors are listed below, and their locations on Alice are shown in Figure 3.2.

- *Sick LADARs.* These were (plane scanning) ranging sensors. Alice was equipped with seven units; five on the front bumper, one on the rear bumper and one on a roof-mounted pan-tilt unit.

- *Riegl LADAR.* The concept was the same as that utilized by the Sick LADAR. However, the data return of the Riegl contained not only distances- but also intensity information. This was used to detect lane lines, curbs and ground structure, in order to determine the geometry of the travelled road segment.
- *Radar.* Alice was equipped with two radars; both mounted on pan-tilt units. Their main purpose was long range detection of moving vehicles. Consequently, they were used mainly in intersections.
- *Bumblebee Stereo vision cameras.* The Bumblebees were integrated stereo vision camera pairs, directed towards the ground. They were used mainly for lane- and stop line detection.
- *Scorpion Stereo vision camera pairs.* Pairs of Scorpion stereo vision cameras were used for obstacle detection, mainly at longer range (∼10 m).

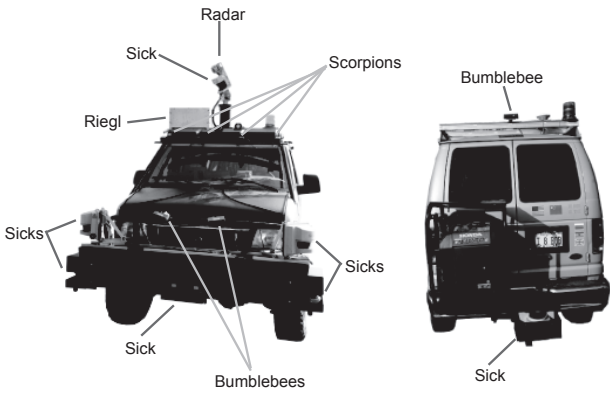



**Figure 3.2** Photograph of Alice, indicating the location of its sensors.

### Computing

The computers used at the NQE were 10 dual core cPCI blades with 1 Gbit Ethernet, one quad core server and two additional dual core machines (left from the 2005 DGC race setup) networked into a decentralized Linux cluster. Some of the machines were connected to consoles via a Keyboard Video Mouse (KVM) switch. The computer rack of Alice is shown in Figure 3.3.

### Actuation

The working principles of the brake-, steering-, transmission- and throttle actuators are described below. The first three are shown in Figures 3.4(a), 3.4(b), 3.4(c), respectively. (The throttle actuator is not depicted, since throttle was actuated by wire in the Ford E-350.) A Programmable Logic Controller (PLC) constituted the interface between one of the cPCI blades and the actual actuators.

- *Throttle.* In the stock E-350 van, the position of the accelerator pedal was read by a sensor, which actuated the Powertrain Control Module (PCM) of the car. When in autonomous mode, the PCM got the actuating signal from the PLC, rather than the above mentioned sensor, allowing the use of the factory built in throttle actuator.
- *Brake.* The brake actuator consisted of a compressor connected to 4 pneumatic pistons via PLC-actuated 'on/off' valves. Consecutive pistons were able to exert forces at a ratio of 1:2 on the actual brake pedal, yielding 16 equally spaced

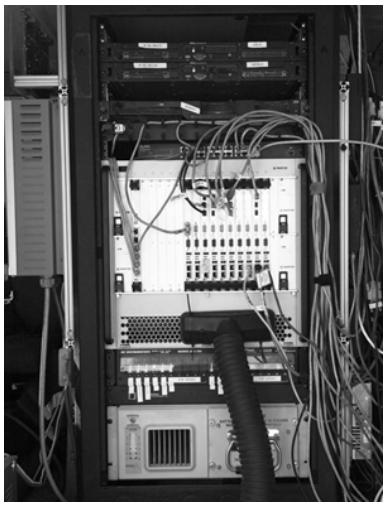
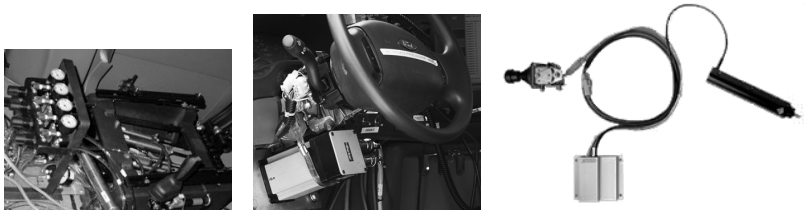


**Figure 3.3** Photograph showing the computer rack of Alice. From top down: core duo machines, quad core machine and cPCI box. (The lowermost box is a battery pack.)



(a) Brake actuator. (b) Steering actuator. (c) Transmission actuator.

**Figure 3.4** Photograph showing the actuators of Alice.

force states. In addition to the main brake, there was a backup system, which was actuated if an emergency stop signal was issued by DARPA.

- *Steering.* Steering was actuated by a DC servo motor (the yellow box in Figure 3.4(b)) connected to the steering column via a chain drive. There were two closed loops – one over the DC servo motor and one over the factory mounted power steering servo. The result of these is further discussed in Section 5.3.
- *Transmission.* The vehicle was equipped with an automatic transmission gear box, but still needed a way to shift between 'park', 'reverse', 'neutral' and 'drive'. This was solved by utilizing an electric linear servo, connected to the PLC.

The dynamics of the brake-, steering- and throttle actuators were of special interest, since they constituted parts of the system to be controlled. They are not explicitly analyzed here. However, they are implicitly captured in the results presented in Section 5.1.

## 3.2 Sensing

As shown in Figure 3.2, Alice was equipped with a large number of sensors. They were set up in a modular fashion, enabling easy reconfiguration or addition of sensors.

The goal of the sensing system was to collect, classify, fuse and finally put data in a map database, which was used by the navigation system to make decisions. The data flow from sensors to map is shown in Figure 3.5.

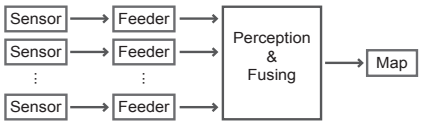



**Figure 3.5** Block diagram illustrating the data flow from sensors to map.

The feeders were basically device drivers, establishing an interface between the hardware and the perceptors. The purpose of the perceptors was to classify certain objects from the sensor raw data streams. These objects included lane lines, stop lines, curbs, the road, static obstacles and moving obstacles (i.e. other vehicles). The problem of segmenting raw sensor data into objects and classifying them, using the perceptors, was augmented with the problem of fusing data from the different sensors. It was important not to erroneously introduce the same object several times into the map only because it was seen by several sensors. Parts of the sensing system are covered more in depth in [28, 13].

## 3.3 Navigation

The purpose of the navigation system was to generate and track trajectories given the MDF, RNDF and information from the map. This complex problem was overcome by decomposing it into smaller, confined problems and implementing software modules which solved them. These software modules formed the navigation stack shown in Figure. 3.6.

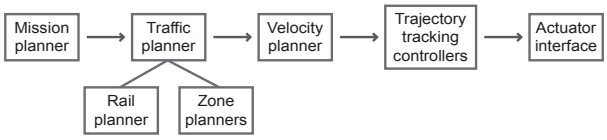



**Figure 3.6** Block diagram illustrating the navigation stack of Alice.

The mission planner generated an ordered sequence of waypoints, corresponding to road segments to traverse in order to complete the MDF. This was done by means of a graph search. The list of waypoints generated by the mission planner was passed to the traffic planner. Depending on the geometry of the environment (road or unstructured zone, e.g. parking zone) either of two traffic planners were invoked:

- *Rail planner.* This was the planner used for nominal road region planning. In each cycle it conducted an A* search over a graph, which was generated offline from the RNDF. (The A* algorithm is a heuristic extension of the classic Dijkstra graph search algorithm.) One of the main challenges here was to associate costs to the individual nodes properly, with respect to the A* algorithm.

- *Clothoid planner.* The clothoid planner was the one out of three competing zone planners, which was chosen for the NQE. It was used in parking- and obstacle zones, as well as road regions where the rail planner failed to plan around obstacles. Trajectories were produced by searching over a tree of pre-generated clothoids, in a cost landscape provided by the map.

The output of either traffic planner was a spacial path. A velocity planner (actually speed planner) populated the points along this path with reference speeds and accelerations before it was sent down to the trajectory tracking controllers which are the subject of the remainder of this thesis.

As opposed to the sensing system, which incorporated a one way data flow (with minor exceptions internal to the perceptors), the navigation system required feedback at several stages. The traffic planner had to communicate to the mission planner in case of road blocks, or other 'failures'. The controllers were able to fail to the traffic planner if an unfeasible trajectory was received (and for some additional reasons mentioned in Section 7.8). Obviously, the controllers made internal use of feedback. Finally, there were two cascaded loops in the steer actuator and power steering servos.

The loops below the traffic planner are further discussed in Chapter 5, with focus on the controller loops. The loops above the controllers were hard to analyze (rigorously) because of their event-based nature, making it difficult to guarantee deterministic behavior in the navigation stack at all times.

# 4. Problem Formulation

This chapter outlines the vehicle control problem, which needed to be approached and solved during the project. First, it was decomposed into the confined sub-problems, listed below.

**System Identification**

The first part of the work was to deduce a parametrized model for the longitudinal- and lateral dynamics of a road vehicle. The parameters needed to be matched to those of the actual vehicle, by means of system identification experiments. The outcome of these experiments would prove if the suggested parametrization was well suited, or if the model needed to be altered.

**Controller Design**

A lateral- (steering-) and a longitudinal (speed) controller needed to be designed and implemented. These controllers should take a trajectory defined by a sequence of points in the ground plane and an associated speed profile, and generate control signals issued to the steering-, gas-, brake- and transmission actuators. Simple algorithms were desired, since they would facilitate the entire development chain, from analysis and implementation to debugging and tuning.

There were no absolute performance requirements. Instead, the goal was to obtain a balance between responsiveness and robustness, which was to be extensively evaluated on the real process, prior to the NQE. Generally, well-damped behavior was prioritized over good performance for high frequencies in the references. This was especially true for the lateral controller.

**Contingency Management**

Apart from performing well during nominal conditions, the controller module required the ability to handle certain special situations, including actuator failures and traffic planner errors, such as extensive delays between trajectories.

**Implementation and Testing**

When the control algorithms and the contingency management scheme were at hand, they needed to be implemented and integrated into the fairly complex system. The controllers needed to receive trajectories from a traffic planner and issue control signals to a software module responsible for hardware actuation. Once implemented, the module needed to be tuned and tested extensively together with the system, to find and correct unexpected behavior.

**Time frame**

The perhaps hardest problem during the project was its tight time frame. It was especially true for the low level controllers since they needed to function adequately in order to test higher level parts of the project, such as traffic planning. This further stressed the need of implementation- and tuning-wise uncomplicated control schemes.

# 5. Modelling and Controller Design

The longitudinal and lateral dynamics of the vehicle are introduced in Section 5.1. This is followed by a description of the longitudinal and lateral controllers in Sections 5.2,5.3. The focus lies on the lateral control problem, which is further investigated in Chapter 6.

## 5.1 Vehicle Dynamics

In order to successfully design and analyse the trajectory tracking controllers, models of the vehicle dynamics were needed. These models are the subject of this section.

### Longitudinal Dynamics

The brake and throttle actuators both acted on the acceleration of the vehicle. In Alice an acceleration control signal $u_a \in [-1,1]$ was utilized. Signals in $[-1,0[$ actuated the brake, whereas $u_a \in ]0,1]$ corresponded to the throttle. Because of dynamics in the power train, ground friction and air resistance, the acceleration of the vehicle was not a strictly linear function of $u_a$. Series of open loop experiments were conducted in order to analyze the non-linearity. During all these experiments the vehicle was driving forward along a straight line on a flat asphalt surface while time stamped speed measurements were written to file. The experimental procedure was as follows:

1. A logger which recorded vehicle speed $v$ , acceleration $a$ and applied longitudinal control signal $u_a$ was started.
2. Subsequently, a constant control signal $u_a$ was issued until the vehicle reached a steady state speed. (Note that the nature of the automatic transmission required slight brake pressure to be applied in order to keep the vehicle stationary.)
3. A step of parametrized height and duration was introduced in $u_a$.
4. When the vehicle speed converged after the step, the experiment was terminated by stopping the logger and the vehicle.

A large number ($\sim$100) of open loop experiments were conducted with different step heights and durations. The results were compiled into a parametrized lookup table, mapping vehicle speed $v$ and desired acceleration $a_u$ to required control signal $u_a$. A 3D representation of the resulting lookup table is shown in Figure 5.1.

To confirm the tacit time-invariance assumption, which was necessary for the validity of the non-linearity model, a subset of the open loop experiment was repeated after a few days.

No explicit attention was given to the non-linearity while the vehicle was reversing. Since the above deduced model constituted a good approximation – at least when reversing at low speeds. This was adequate, since there were no plans to reverse autonomously at high speeds.

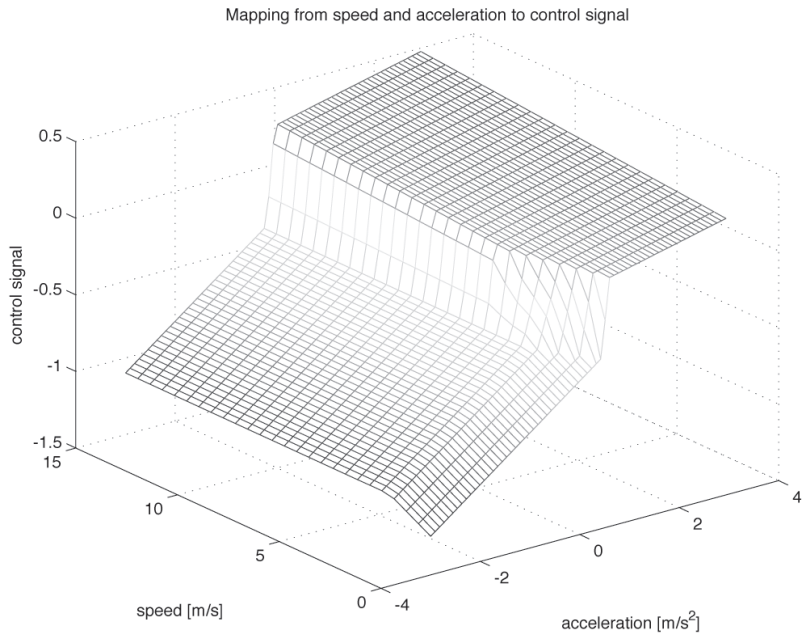



**Figure 5.1** 3D representation of the parametrized lookup table mapping $(v, a_u) \rightarrow u_a$.

### Lateral Dynamics

The vehicle was assumed to have Ackermann steer dynamics, which enabled the use of a bicycle model approximation. Turning radius $r$ and front wheel angle $\phi$ were related through $\tan \phi = \frac{L}{r}$, where $L = 3.55$ m was the vehicle wheel base. Since the vehicle was supposed to drive mainly on paved roads at relatively low speeds, no effort was made to model wheel forces in order to detect and prevent slippage. consequently, $L$ was the only parameter in the model of the lateral dynamics.

As opposed to the longitudinal control problem, where the error metric was chosen to be the trajectory reference speed subtracted by measured speed, there was no obvious error metric associated with the lateral control problem. It was, however, natural to decompose the error into two components:

- *Distance error.* A distance $e_y$ between a point $P$ on the vehicle and a reference point $R$ on the trajectory.
- *Angle error.* An angle $e_\theta$ between the heading of the vehicle and the tangent of a reference point on the trajectory.

The point $P$ on the vehicle was chosen to be the center of the rear axle. It will be shown later in this section that the particular choice enabled the proposed lateral control strategy to be applied also for reverse operation of the vehicle (with a slight modification). The reference trajectory point $R$ was chosen as the orthogonal projection of $P$ onto the trajectory (i.e. the trajectory point closest to the rear axle center).

Assuming that the reference trajectory was a circle of radius $r_c$, as shown in Figure 5.2, the lateral dynamics were given by (5.1, 5.2). The circular trajectory assumption was fair, since an arbitrary feasible trajectory is locally well approximated by a circle arc.

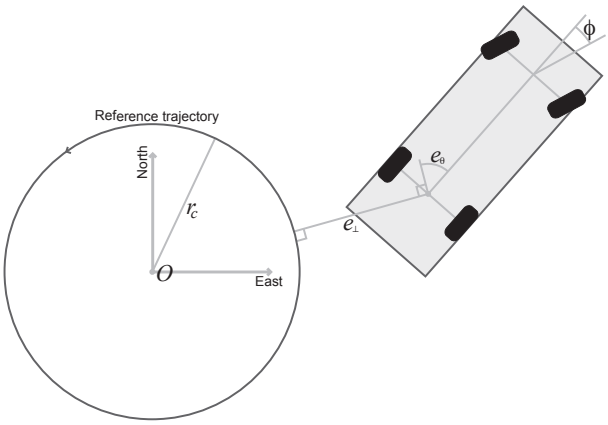



**Figure 5.2** Figure used in the deduction of (5.1, 5.2).

$$\frac{de_y}{dd} = \sin e_\theta \tag{5.1}$$

$$\frac{de_\theta}{dd} = \frac{\cos e_\theta}{e_y + r_c} + \frac{\tan \phi}{L}. \tag{5.2}$$

Note that the propagation of the error state $(e_y, e_\theta)$ in (5.1,5.2) is given over travelled distance $d$ , rather than time. This simplification is equivalent to assuming constant vehicle speed,$v$, and hence decreasing the degree of the system by one. It will be used throughout the analysis of the lateral controller and its usefulness becomes evident in Section 6.1, when the lateral controller is analysed by means of phase portraits.

In Alice, the output of the lateral controller was the set point for a PID loop within the steer servo motor. The steer servo applied torque to the steer column, as would a driver using the steering wheel. It was further aided by the factory built-in power steering of the Ford E-350, which constituted another closed loop servo system. In order to protect the servo motor gear box, a rate limiter on $\dot{\phi}$ was implemented, according to (5.3). The maximal steer rate $\dot{\phi}_{\max}$ was gain scheduled with vehicle speed, through a constant $C_0$ .

$$\dot{\phi}_{\max} = C_0 |v| \tag{5.3}$$

The dependency on $v$ in (5.3) was debatable. The ultimate purpose of the rate limiter was not to limit $\dot{\phi}$, but rather the torque on the steering column. Since there was no hardware in the loop which could communicate the steer column torque to the rate limiter software, this would involve predicting the torque from other measurements. The steer column torque depended critically on at least vehicle speed, steer angle and brake pressure, turning its prediction into a fairly hard modelling problem and explaining the simplistic approach of (5.3).

Rather than explicitly dealing with the rate limiter dynamics when designing the lateral controller, the speed planner was made responsible of generating speed profiles which were feasible with respect to the spatial part of the reference trajectories and the steer dynamics.

Apart from the rate limiter dynamics 5.3 there was a pure delay $\tau_\phi$ between issued control signal $u_\phi \in [-1, 1]$ and response in $\phi$. By means of step response experiments, the delay was determined to be $\tau_\phi = 400$ ms and could therefore not be neglected.

## 5.2 Longitudinal Control Strategy

After the introduction of the linearizing lookup table in Figure 5.1, it was straight forward to design a speed profile tracking controller, using any standard linear design method.

A PI controller was chosen, as opposed to the PID controller used for the 2005 DGC. In fact, the D-part would have added functionality, since speed was controlled by actuating acceleration directly.

The PID controller used for the 2005 DGC did not utilize a linearizing lookup table. This resulted in either a bunny-hopping or overshooting behavior when the vehicle stopped at stop lines etc. The mechanism behind the bunny-hopping was:

1. The vehicle braked too hard and stopped some distance before the stop line.
2. The integral built up and eventually put the vehicle back in motion.
3. The above was repeated until the stop line was reached (or passed by at most one 'hop length').

Using the lookup table, the bunny-hopping was suppressed, but not entirely eliminated. There was a discussion whether to implement a position (as opposed to speed) controller and switch to it when the vehicle was to stop at a certain point. However, this would have required the implementation of a bumpless transfer mechanism and an augmentation of the trajectory structure. In order to generate a position error, the trajectory points would need to be associated with a reference passing time. It would hence be necessary to calculate and update these times from the speed profile of the trajectory during run time. Eventually it was decided that the potentially gained functionality (smoother stops) did not justify the increased complexity – and the development time it would take to obtain it.

Apart from the longitudinal dynamics captured by the lookup table in Figure 5.1 there was a constant delay $\tau_a$ in actuation of the throttle and brake. It corresponded to a travelled distance $d_{FF,a}$ through (5.4), where $\hat{\tau}_a$ was an estimate of the acceleration actuation delay.

$$d_{FF,a} = v\hat{\tau}_a \tag{5.4}$$

The speed planner populated the trajectory points with both reference speeds $v_r$ and corresponding reference accelerations $a_r$ . A prediction term consisting of a constant times the reference acceleration $a_{FF}$ corresponding the point a distance $d_{FF,a}$ in front of $R$ along the trajectory was augmented to the PI control law in order to suppress the effects of the actuation delay. The augmented control law, with speed error integral $I_v$ and parameters $C_1, C_2, C_3$ was given by (5.5).

$$a_u = C_0(v_r - v) + C_1 a_{FF} + C_2 I_v \tag{5.5}$$

Integral anti-windup was implemented on $C_2 I_v$. The idea was to limit the integral, so that it would not result in control signals $u_a$ outside $[-1, 1]$. An analytic solution to this problem was prevented by the form of the linearizing function shown in Figure 5.1. Instead, the nominally updated integral was stepwise increased/decreased until (5.5) yielded $u_a \in [-1, 1]$. Later on the feed forward term $C_1 a_{ff}$ was removed from (5.5) when checking for $u_a \in [-1, 1]$, preventing the integral from changing rapidly when there was a large change in feed forward acceleration. This was a justified change, since the role of the integral (capturing unmodelled low frequency dynamics) was not affected.

PI control of a first order process is perhaps one of the most explored areas of automatic control. Consequently, no further analysis of the longitudinal controller will be given, in favor of the lateral control strategy to be presented in Section 5.3.

## 5.3 Lateral Control Strategy

The lateral control problem was more difficult than the longitudinal one, treated in Section 5.2. The main reason being the error dynamics (5.1,5.2). Rather than implementing a linear controller, geometric reasoning was used to obtain an intuitively appealing strategy, described below.

### The Existing Solution

In the 2005 DGC, the lateral control of Alice was governed by a PID controller. The control error was a linear combination of the cross track- and yaw errors. As shown in (5.1,5.2), the system with control signal $\phi$ and output $e_y$ was nonlinear. Experience from the 2005 DGC showed that the PID controller had difficulties in handling the particular nonlinearity. Consequently, a new lateral control approach was sought.

Before continuing it is, however, worthwhile mentioning that the lateral control problem in nonholonomic road vehicles has been subject to numerous approaches, including [29, 24, 30, 11, 23, 20, 32]. Studying these and conclusions from their implementers constituted a useful background and helped in avoiding some pitfalls.

### The New Lateral Controller

The design to be presented below was based on geometric reasoning and yielded an inherently stable nonlinear state feedback controller with $l_2$ as its only tunable parameter. Further, its geometric nature facilitated realtime visualization and debugging, as mentioned in Section 7.9.

The control strategy is easily explained, using Figure 5.3. The rear axle center of the real vehicle was projected onto the reference trajectory, as described in Section 5.1. The heading of the arising *virtual* vehicle was aligned with the tangent of the trajectory at $R$. Subsequently, the front wheels of the virtual vehicle were turned so that its turning radius coincided with the curvature of the reference trajectory at $R$. The real vehicle was given a front wheel angle $\phi$, steering it towards the end of an imaginary handle of length $l_2$, pulling the virtual vehicle along its path.

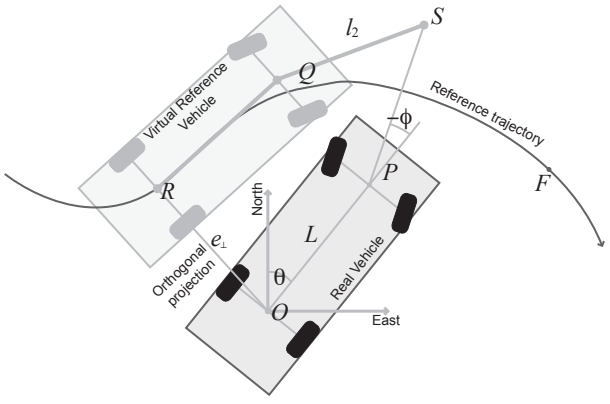



**Figure 5.3** Graphical illustration of the lateral control strategy.

Since the system was subject to an actuation delay $\tau_\phi$, as discussed in Section 5.1, the steering angle of the virtual reference vehicle was not computed from the curvature of the reference trajectory at $R$, but rather its curvature a distance $d_{FF,\phi}$ ahead of $R$.

### An Extension for Reversing

Exploiting symmetry and the error metric introduced in Section 5.1, the controller was easily modified for reverse trajectory tracking. This was done by mirroring the (real) vehicle through its rear axle and applying the control strategy to the mirrored vehicle, as shown in Figure 5.4.

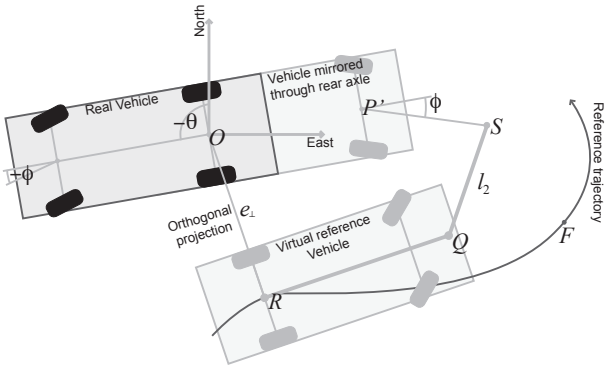



**Figure 5.4** Illustration showing the extension of the lateral control strategy to the case of reverse driving.

### Integral Action and Anti-windup

The lateral controller was augmented with an integral part which was added to the nominal control signal, to form the final steering angle $\phi$. The integral update law was given by (5.6).

$$\frac{dI}{dt} = \frac{[e_y(t) + C_0 \cdot \sin(e_\theta(t))]\, v(t)^{C_1}}{C_2} \tag{5.6}$$

The error metric in (5.6) was equivalent to measuring the perpendicular error of the vehicle's center line, a distance $C_0$ in front of $P$. In order to integrate the error over travelled distance, $C_1 = 1$ was chosen. However, this introduced oscillations for high speeds. It was empirically found that reducing the scaling power to $C_1 = 0.5$ removed the oscillations.

The power steering PID loop successfully depressed load disturbances. Hence, the only role of the integral action was to account for miscalibrations in steer angle measurement. It was possible to make the integral slow enough not cause noticeable overshoots, because of the low frequency nature of the miscalibration.

Given the slow update time and small operational values, the only anti-windup action taken, was to confine the lateral integral to the interval $[I_{\min}, I_{\max}]$, with empirically adjusted bounds.

# 6. Controller Analysis

The nonlinear state feedback lateral controller is analyzed in this chapter. In Section 6.1 the global behavior of the controlled system is investigated by means of phase portraits. Section 6.2 introduces a linearization of the controlled system around the zero error equilibrium, which is the region of nominal operation. The model is used to investigate stability margins and suggest a gain schedule of the controller parameter $l_2$. Finally, global stability is discussed in Section 6.3.

## 6.1 Phase Plane Analysis

In order to analyze the performance of the lateral controller, the evolution of its error state $(e_y, e_\theta)$ was investigated by means of phase portraits. As suggested in Section 5.1, velocity dependence was dropped by plotting the error state evolution over travelled distance $d$, rather than time.

Ultimately, it was desired to investigate the evolution of the error states with the reference trajectory taken to be an arbitrary parametrized curve in the ground plane. However, this was not possible, since $(\frac{\mathrm{d}e_y}{\mathrm{d}d}, \frac{\mathrm{d}e_\theta}{\mathrm{d}d})$ at a point $(e_y, e_{\theta,0})$ would not be uniquely defined, but depend on the location along the reference trajectory. The problem was resolved by using circular trajectories and corresponding error dynamics (5.1,5.2) derived from Figure 5.2 in Section 5.1. Circular reference trajectories were chosen, since they were the most general curves, not leading to uniqueness problems in $(\frac{\mathrm{d}e_y}{\mathrm{d}d}, \frac{\mathrm{d}e_\theta}{\mathrm{d}d})$. They were also good local approximations to any feasible trajectory.

Further, model- and measurement errors were neglected and integral action was omitted. The front wheel angle $\phi$ was confined to $[\phi_{\min}, \phi_{\max}] = [-0.45 \text{ rad}, 0.45 \text{ rad}]$, with numerical values obtained from measurements on Alice. Since curvature was constant along the circular trajectory, the effect of the steering actuation delay $\tau_\phi$, introduced in Section 5.1 could be neglected.

Under the above circumstances, the control law was given by (6.1,6.2,6.3) where $\phi_v$ was the front wheel angle of the virtual vehicle and $r_c$ was the radius of the circular reference trajectory. Here, $\delta$ was the angle between the trajectory tangent at $R$ (see Figure 5.2) and steering wheel heading of the real vehicle. (The function $\mathrm{atan2}(x,y)$ in (6.2) is equivalent to $\arctan(\frac{x}{y})$ with domain $\mathbb{R}\backslash(0,0)$ and extended range $[-\pi, \pi]$.)

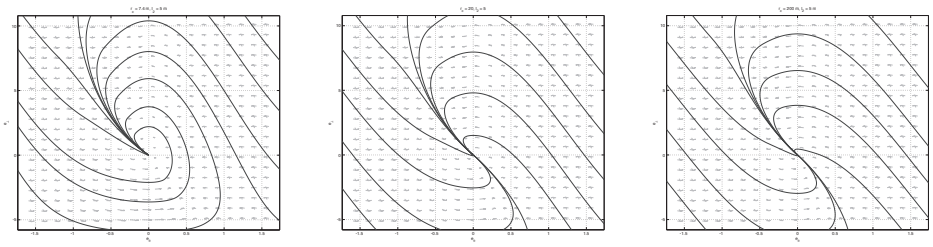
$$\phi_v = -\arctan\frac{L}{r_c} \tag{6.1}$$

$$\delta = \mathrm{atan2}(l_2 \sin\phi_v - L\sin e_\theta - e_y, L + l_2\cos\phi_v - L\cos e_\theta) \tag{6.2}$$

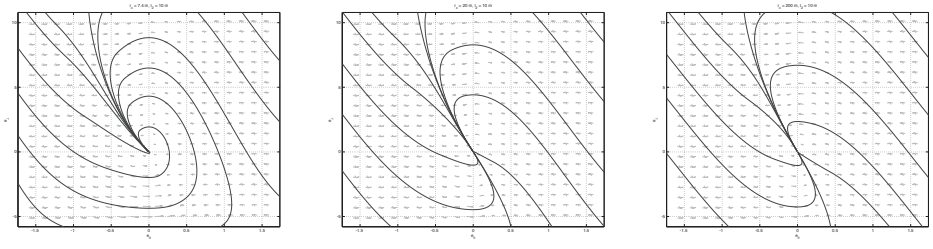
$$\phi = \mathrm{sat}_{\phi_{\min},\phi_{\max}}(\delta - e_\theta) \tag{6.3}$$

Figure 6.1 shows a family of phase portraits drawn under the conditions described above with $r_c \in \{7.4, 20, 200\}$ [m], $l_2 \in \{5, 10, 20, 30\}$ [m]. As seen in the phase portraits, the origin was a stable equilibrium.

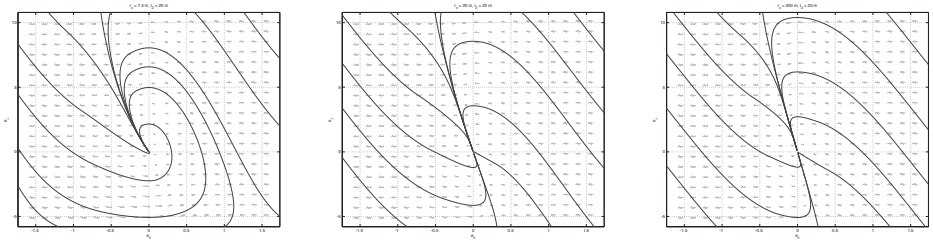
When tracking straight trajectories, steer authority was symmetric with respect to $e_y = 0$ m. Tracking a circular trajectory of constant turning radius $r_c < \infty$ m required a nominal steer angle $\phi \neq 0$ rad, yielding an asymmetric division of steer authority between the cases $e_y < 0$ m and $e_y > 0$ m. Accordingly, phase portraits in Figure 6.1 show that $e_y > 0$ m were handled more efficiently as $r_c$ decreases, whereas $e_y < 0$ m required longer distances to eliminate for small $r_c$.



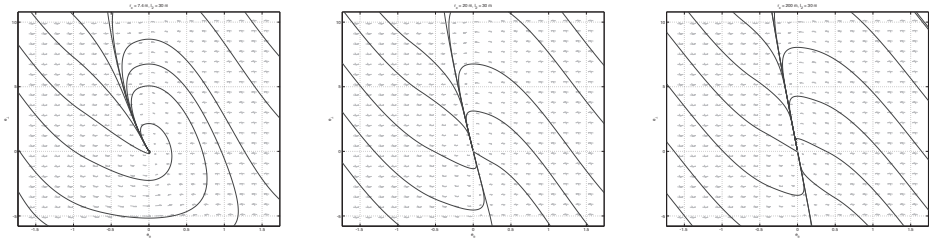
(a) $r_c = 7.4\ m, l_2 = 5\ m$ (b) $r_c = 20\ m, l_2 = 5\ m$ (c) $r_c = 200\ m, l_2 = 5\ m$



(d) $r_c = 7.4\ m, l_2 = 10\ m$ (e) $r_c = 20\ m, l_2 = 10\ m$ (f) $r_c = 200\ m, l_2 = 10\ m$



(g) $r_c = 7.4\ m, l_2 = 20\ m$ (h) $r_c = 20\ m, l_2 = 20\ m$ (i) $r_c = 200\ m, l_2 = 20\ m$



(j) $r_c = 7.4\ m, l_2 = 30\ m$ (k) $r_c = 20\ m, l_2 = 30\ m$ (l) $r_c = 200\ m, l_2 = 30\ m$

**Figure 6.1** Phase portraits showing the state evolution of error states $e_y, e_\theta$ for different values of the trajectory radius $r_c$ and controller parameters $l_2$.

Another observation was that controllers with larger $l_2$ were more efficient in aligning the vehicle with the reference trajectory, but less efficient in eliminating the cross track error. This was obvious from the waggon-pulling analogy presented in Section 5.3.

Figure 6.2 illustrates what happened if the radius of the reference trajectory was smaller than the minimum turning radius of the vehicle. The phase portrait was generated with $r_c = 5$ m, while the minimum turning radius of Alice was 7.4 m. Consequently, the state $(e_y, e_\theta) = (0,0)$ was no longer an equilibrium point and there was instead a limit cycle around it.

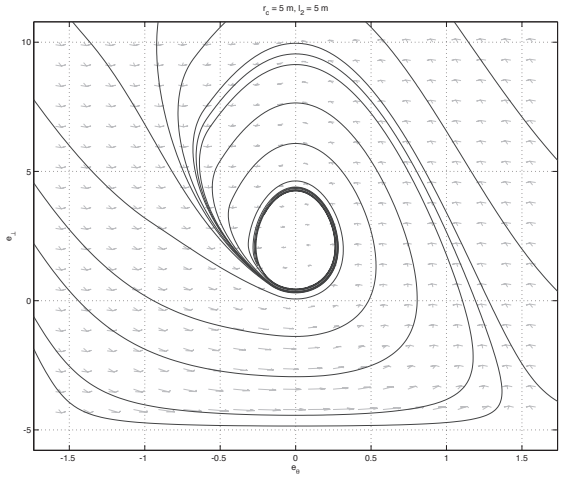



**Figure 6.2** Phase portraits illustrating tracking of a circular trajectory with radius $r_c = 5$ m, which was smaller than the minimum turning radius of Alice (7.4 m). The controller parameter was set to $l_2 = 5$ m.

Although $e_\theta \leq \pm\pi$ rad was true during all nominal operation, the global behavior, with initial $e_\theta \in [-\pi, \pi]$, was investigated through further phase portraits. A representative example is shown in Figure 6.3.

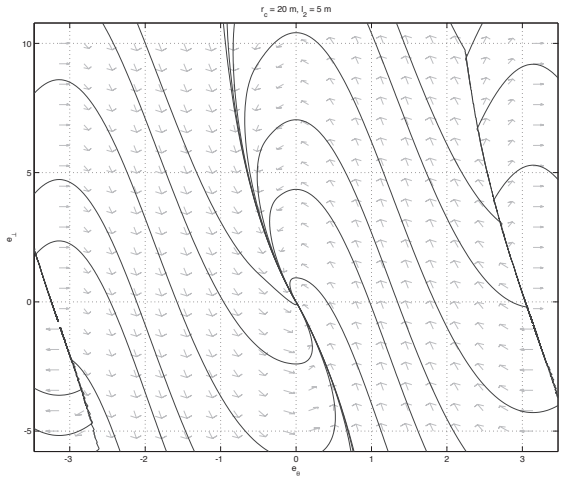



**Figure 6.3** Phase portrait illustrating the global behavior of the controlled system while tracking a circular trajectory with radius $r_c = 20$ m. The controller parameter was set to $l_2 = 5$ m.

When the heading of the vehicle made an angle $e_\theta = \pm\pi$ rad with the tangent of the reference trajectory, while the cross track error was $e_y = 0$ m, the control law could not decide whether to issue control signal $u_\phi = -1$ or $u_\phi = 1$ in order to turn the vehicle around. For cross track errors $e_y \neq 0$ m, the same thing occurred for small $e_\theta \neq 0$ rad. These equilibrium points formed the near vertical lines at $e_\theta \approx \pm\pi$ rad in Figure 6.3. In practice, these unstable equilibria could be ignored since they were unstable and not reached in practise.

## 6.2 Linearization and Gain Scheduling

Since the nominal operation of the controller was in a region of small $(e_y, e_\theta)$ and large $r_c$, extra attention was given to this region. The closed loop system was linearized around $(e_y, e_\theta) = (0,0)$ as $r_c \to \infty$ m.

The perpendicular error $e_y$ was decomposed into a perpendicular reference $y_r$ and position $y$ . Correspondingly, $e_\theta$ was replaced by $\theta_r$ and $\theta$. However, $\theta_r$ was omitted since $r_c \to \infty$ m $\Rightarrow \theta_r = 0$ rad. The hereby resulting state update equation and control law were given by (6.4,6.5), respectively.

$$\begin{bmatrix} \frac{dy}{dd} \\ \frac{d\theta}{dd} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}}_{A_o} \begin{bmatrix} y \\ \theta \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ \frac{1}{L} \end{bmatrix}}_{B_o} \phi \tag{6.4}$$

$$\phi = -\frac{1}{l_2}(y - y_r + \theta L) - \theta \tag{6.5}$$

Combination of (6.4,6.5) resulted in the closed loop system (6.6).

$$\begin{bmatrix} \frac{dy}{dd} \\ \frac{d\theta}{dd} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 \\ -\frac{1}{l_2 L} & -\frac{l_2+L}{l_2 L} \end{bmatrix}}_{A_c} \begin{bmatrix} y \\ \theta \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ \frac{1}{l_2 L} \end{bmatrix}}_{B_c} y_r \tag{6.6}$$

$$y = \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{C} \begin{bmatrix} y \\ \theta \end{bmatrix}$$

The closed loop transfer function $G$ from $y_r$ to $y$ was now given by (6.7). In (6.8), the transfer function $G_o$ of the open loop system is obtained.

$$G(s) = C(sI - A_C)^{-1} B_C = \frac{1}{l_2 L s^2 + (L + l_2)s + 1} \tag{6.7}$$

$$G(s) = \frac{G_o(s)}{1 + G_o(s)} \Rightarrow G_o(s) = \frac{G}{1 - G} = \frac{1}{L s^2 + (L + l_2)s} \tag{6.8}$$

The cutoff 'frequency' $\omega_c$ [rad m$^{-1}$] was now easily obtained from (6.8) through $|G_o(i\omega_c)| = 1$ and plotted as a function of $l_2$ in Figure 6.4. Figure 6.5 shows the phase margin $\phi_m = \pi + \arg G_o(i\omega_c)$ [rad] as a function of the controller parameter $l_2$.

It should be kept in mind that the cutoff 'frequency' $\omega_c$ varied with $l_2$, making direct interpretation of the phase margin plot in Figure 6.5 difficult. It was more interesting to evaluate the delay distance margin $\tau_m = \frac{\phi_m}{\omega_c}$ [m] as a function of $l_2$. This was done, with result shown in Figure 6.6.

Motivated by Figure 6.6, the gain schedule $l_2(v) = C_0 v$ was introduced. Different values of $C_0$ were evaluated both in simulation and the field, upon which $C_0 = 2.0$ was chosen.

The increase of $l_2$ with $v$ was also intuitively appealing, considering steer gain interpretation of $l_2$, introduced in section 5.3. With increased vehicle speed, the system would require a smaller steering gain in order to maintain its error decrease rate $(\frac{de_y}{dd}, \frac{de_\theta}{dd})$.

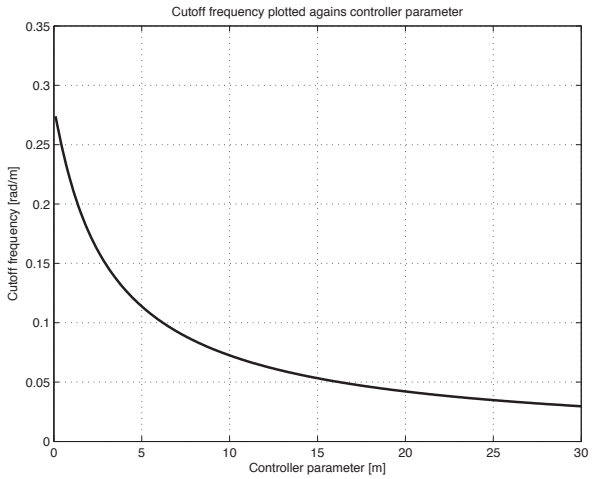



**Figure 6.4** Cutoff 'frequency' $\omega_c$ [rad m$^{-1}$] as function of $l_2$.

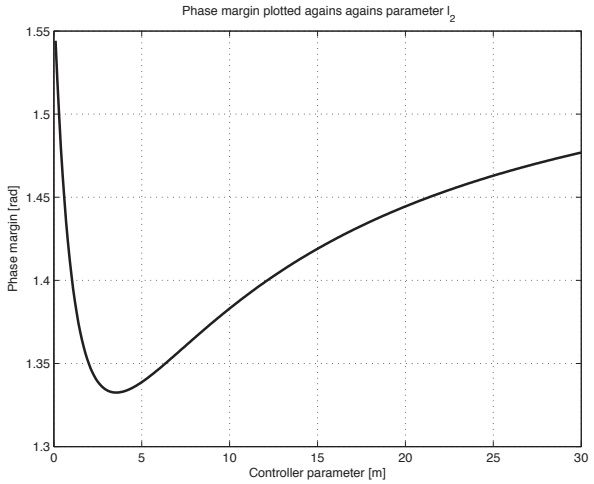



**Figure 6.5** Phase margin $\phi_m$ as function of $l_2$.

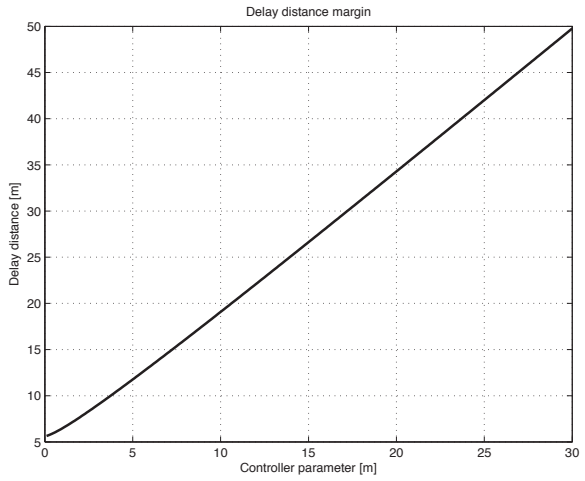



**Figure 6.6** Delay distance margin $\tau_m$ [m] as function of $l_2$.

## 6.3 Global Stability

The analysis in Section 6.2 indicated stability with adequate margins around the zero error equilibrium. Further, geometric reasoning in Section 5.3 and the phase portraits in Section 6.1 suggested global asymptotic stability (with the exception for the unstable equilibrium points mentioned in Section 6.1). This was also validated, both in simulation and the real process. However, devising a proof of asymptotic stability was not an obvious problem to solve. A few Lyapunov-like approaches were made, but unfortunately this research was prevented by the tight time frame of the project before a proof was obtained.

# 7. Implementation

This chapter deals with the implementation of the trajectory tracking controller module. In Section 7.1 the structure, inherited by all software modules in the navigation stack, is explained. Section 7.2 reviews additional coding practise. This is followed by a discussion about the usage of threads in Section 7.3. The trajectory class is explained in Section 7.4. The actual implementation of the closed loop controllers is then handled in Section 7.5, with special attention to inter-module communications given in Sections 7.6 and 7.7. Finally, contingency management and the user interface are discussed in Sections 7.8 and 7.9, respectively.

## 7.1 Canonical Software Structure

All software modules in the navigation stack (including the trajectory tracking controller) adhered to the Canonical Software Structure (CSS) framework. The original purpose of this framework was threefold:

- *Modularity.* Providing flexibility and ease of re-use.
- *Consistent structure.* Making code more readable.
- *Standardization.* Providing predictability and increasing robustness.

The CSS structure standardized the interface between modules as well as their internal structure and was implemented by inheriting from an abstract class. It also provided generic utilities for logging to files. The components of a CSS module are shown in Figure 7.1 and described below.

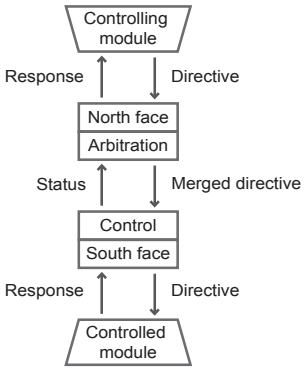



**Figure 7.1** Block diagram illustrating the structure of a CSS software module.

### North Face

The module implementing CSS, here referred to as the control module, received directives through a queue-buffered north face towards a controlling module. (The CSS allowed a module to have several north faces, if needed). In the case of the trajectory tracking controller module, the directives were the trajectories to be tracked. The control module replied with responses to the received directives over the north face. For the trajectory tracking controller, these were status messages used for contingency management, which is discussed in Section 7.8.

### Arbitration

Receiving directives and sending responses was handled by the arbitration block. It's main purpose was to accept or reject incoming directives and send proper responses, depending on the state of the control module.

### Control

The arbitration block communicated with the control block by sending merged directives and receiving status messages. (The term merged was explained by the case when the control module had more than one north face.) The control block was where the actual functionality of the control module lived. (In the CSS context control had nothing to do with controller.) Apart from merged directives, the control block could also have ports towards estimators and device drivers and a south face towards the controlled module (or several south faces towards controlled modules). The control block made use of data arriving from arbitration and hardware ports, and generated directives which were sent to the controlled module over the south face(s).

### South Face

The south face was the interface between the control module and the controlled module. Structurally, the north and south faces were very similar.

## 7.2 Coding Practice

The CSS constituted a framework, defining code structure on a module level. In this section additional implementation objectives, held in mind while implementing the trajectory tracking controller module, will be reviewed.

### Dynamic Memory Allocation

Dynamic memory allocation 'in the loop' is slow and also dangerous, since it easily leads to segmentation faults caused by memory leaks or dangling pointers. Whenever possible, memory was allocated at program initialization, and deallocated when the program terminated.

### Inheritance

Inheritance, especially multiple inheritance, easily leads to pitfalls caused by incompatibilities. Because of this, 'has-a' relations were preferred to 'is-a' ditto.

### Threads

Multi-threaded programs are generally harder to debug than single threaded ones. Also, code that is not thread safe easily leads to data corruption, when used in threaded environments. It was therefore desirable to minimize the number of simultaneously running threads. A short discussion is given in Section 7.3.

### Magic Numbers

The occurrence of unexplained numerical values in the code was minimized. Where suitable, constants were defined in parameter files, which could be loaded during runtime. For non-tunable constants, the `#define` and `enum` constructs were used.

### Variable Scope

Limiting the scope of variables and functions was stressed. This increased the modularity of the code, and facilitated debugging.

## 7.3 Threads

As mentioned in Section 7.2 the number of threads was kept at a minimum. However, it was not practical to completely eliminate multi-threading. This was partly because trajectories were sent asynchronously and partly because the Sparrow Hawk user interface (see Section 7.9) started a separate thread. Hence, the threads started by the trajectory tracking controller module were:

- *Trajectory receiving thread.* This thread listened for trajectories on the north face. A mutex was implemented in order not to change reference trajectory during an ongoing control cycle. See Section 7.6 for details.
- *Main program thread.* This thread ran the actual controllers.
- *User interface thread.* The Sparrow Hawk user interface described in Section 7.9 was implemented using a separate thread.
- *Canonical Software Structure.* The inherited CSS structure ran several threads. It was, however, designed to be transparent to modules which inherited it.

## 7.4 The Trajectory Class

This section describes the class which implemented the trajectories to be tracked by the trajectory tracking controller module. After reviewing the original implementation, some augmentations and structural changes are suggested.

### Original Implementation

A class, defining trajectories, was already in place at the beginning of the project. The trajectory object was defined by a trajectory header:

```
struct STrajHeader {
  int m_numPoints;  // number of points in this trajectory
  int m_order;  // max derivs of each point+1 (x,xdot -> 2)
  int m_dir;  // direction of the trajectory (-1 = rev)
} __attribute__((packed)) m_trajHeader;
```

The header defined the number of points in the trajectory, `m_numPoints`. Each trajectory point was assigned Cartesian coordinates $(n,e)$, , defining northing and easting, as well as time derivatives $\{(\dot{n},\dot{e}),(\ddot{n},\ddot{e})...(n^{(m)},e^{(m)})\}$ of these. The maximal order $m$ was defined through `m_order`. Further, the trajectory object was associated with a direction `m_dir` being either forward or reverse. It determined the driving direction of the tracking vehicle.

The actual trajectory was stored in two fixed sized arrays, allocated at construction:

```
double  *m_pN;
double  *m_pE;
```

The size of these arrays was given by `#define TRAJ_MAX_LEN  5000` and constituted an upper bound on `m_numPoints` in the trajectory header. The first `TRAJ_MAX_LEN` elements of the northing array `*m_pN` were northing coordinates, $n$. The next `TRAJ_MAX_LEN` elements, were first order derivatives, $\dot{n}$, and so on. (The same applied for the easting array `*m_pE`.) The derivative of order $p$ associated with point $i$ was hence easily accessible using
`#define INDEX(i,p) ((i) + (p)*TRAJ_MAX_LEN)`.

### Interpolation

Linear interpolation of $\{(\dot{n},\dot{e}),(\ddot{n},\ddot{e})...(n^{(m)},e^{(m)})\}$ was used between consecutive trajectory points, since the lateral controller assumed spatially continuous trajectories.

For bandwidth (and log file size) reasons, it was desirable to maximize the distance between consecutive trajectory points, without significantly losing precision. Given the minimal turning radius $r_{\text{min}} = 7.35$ m for Alice and assuming spatially feasible (i.e. trackable) trajectories, the worst case deviation from a truly continuous trajectory would occur for trajectories with curvature corresponding to $r_{\text{min}}$. This would give trajectories consisting of linear interpolated segments of maximal length $d_i < 2r_{\text{min}}$ a maximal perpendicular deviation $r_{min} - \sqrt{r^2 - \frac{d_i}{2}^2}$ [m] from their continuous counterparts. A trajectory point separation $d_i = 0.5$ m was chosen, yielding a maximal interpolation error of 4.3 mm.

### Suggested Improvements

The ability to specify derivatives of order $m > 2$ in the trajectory points, was superfluous, since acceleration $(m = 2)$ was directly actuated by the throttle and brake.

Further, representing the trajectory points as Cartesian tuples $\{(n,e),(\dot{n},\dot{e}),(\ddot{n},\ddot{e})\}$ may seem appealing. However, the representation $(n,e),\theta_r,v_r,a_r$, where the reference angle $\theta_r$ specifies the reference direction of movement in the $n,e$-plane, $v_r$ is speed reference, and $a_r$ is scalar acceleration reference would be superior for two reasons:

1. The finite (fixed) length of the `double` was limiting when extracting direction information for small values of $(\dot{n},\dot{e})$. Particularly, no direction information could be obtained for $\{(\dot{n},\dot{e}) = (0,0)\}$.
2. Both the lateral controller and traffic planners used the latter representation internally. Consequently, Cartesian trajectory point representation involved two unnecessary conversions, in which precision was potentially lost.

Because of this, a new trajectory class with the suggested improvements was implemented. However, incorporating it into the system would have required changes in several modules and was not prioritized, since there were more urgent issues at other places in the planning stack.

## 7.5 Control Loop

This section reviews the implementation of the CSS module, hosting the controllers. It is done by following the execution of one control cycle. A timing diagram showing the different execution steps of the control loop is shown in Figure 7.2 and explained below. The control- and state update periods were chosen to be 100 ms, which was adequate to capture the significant dynamics of the system (with some margins).

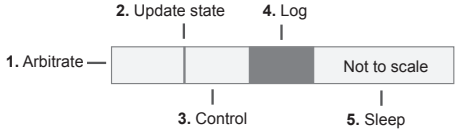



**Figure 7.2** Timing diagram showing one control loop cycle.

### Arbitration

The arbitrate function was invoked at the beginning of each control cycle. Its purpose was to populate and send response messages to the traffic planner through the north face. These responses were populated from a *control status* object, hosting two categories of information. One was the actuator status information, which was populated in the arbitrate function by reading actuator status responses queued at the south face. The other part was populated during the previous control cycle and used for contingency management. See Section 7.8 for a description of the control status object.

### State Updates

After return of the arbitration function, the state variables used by the controllers were updated. They were:

- *Vehicle State.* The vehicle state was read from a separate module, which communicated with the Applanix state estimator described in Section 3.1. It estimated the *position* (northing, easting, altitude) and *orientation* (roll, pitch, yaw) of the vehicle as well as the *first time derivative* of the listed entities. For convenience, the state was given in both (global) Universal Transverse Mercator (UTM) and (local) vehicle frame coordinates. The vehicle state `struct` also held a time stamp used for detecting delays and when logging to file.
- *Actuator State.* The actuator state held information from the steer, throttle, brake and transmission actuators. A status (on/off), commanded position, measured position and time stamp were populated by each mentioned actuator. The actuator state also told whether the vehicle was stopped (used system wide) as well as the E-stop state, explained below. (It also had some deprecated members, not mentioned here.)
- *E-stop State.* Implementation-wise, the E-stop state was part of the actuator state, despite the separate attention given here. It was defined by the following enumeration:

```
enum EstopStatus {
  EstopDisable = 0,
  EstopPause = 1,
  EstopRun = 2
};
```

During nominal autonomous operation the E-stop state was `EstopRun`. The `EstopPause` state was used when the vehicle needed to be momentarily stopped (either by a safety driver or remotely). Transition from `EstopPause` to `EstopRun` was straight forward, whereas transition from `EstopDisable` to `EstopRun` required manual re-initialization of the system. `EstopDisable` was used when the vehicle was manually driven (on e.g. public streets) or at other occasions when spurious transitions into `EstopRun` were unwanted, or even dangerous.

### Control

Once all states were updated, The controllers were invoked in the following order:

- *Longitudinal Controller.*
- *Lateral Controller.*

- *Transmission Controller.*

The controllers were passed pointers to the current trajectory, vehicle state, control status object and actuator directives to be populated. Once all three controllers had returned, the corresponding actuator directives were passed down through the south face to the hardware actuating software module.

**Logging**

During each cycle, the trajectory tracking controller module logged its current state to file. The log files could be considered an augmentation to the user interface and are therefore briefly treated in Section 7.9.

**Sleep**

In order to maintain a constant control period, the process in which the controllers were running was issued to sleep for a variable amount of time at the end of each control cycle. As shown in Figure 7.2 the majority of the execution time (97 %) was spent sleeping.

## 7.6 North Face communication

The trajectory tracking controller CSS module had a north face towards the traffic planner. It was used to receive trajectories and send status responses. The latter was tightly associates with contingency management and is treated in Section 7.8. The code of the trajectory receiving thread is shown below.

```
void Follower::CommThread ()
{  int trajSocket;
  trajSocket = m_skynet.listen(SNtraj, SNplanner);
  CTraj* newTraj = new CTraj(3);
  CTraj* tmpTraj;

  while (!ExitHandler::quit) {
    bool trajReceived = RecvTraj (trajSocket, newTraj);
    while (m_skynet.is_msg(trajSocket)) {
      trajReceived = RecvTraj(trajSocket, newTraj);
    }
    if (trajReceived) {
      m_trajectoryCounter++;
      DGClockMutex(&m_recvdTrajMutex);
      tmpTraj = newTraj;
      newTraj = m_recvdTraj;
      m_recvdTraj = tmpTraj;
      DGCunlockMutex(&m_recvdTrajMutex);
    }
  }
  delete newTraj;
}
```

The purpose of the thread was to copy incoming trajectories to a memory location pointed at by `m_recvdTraj` (which was accessible from the controllers), while locking the `m_recvdTrajMutex` and hence rendering the copy operation atomic, from the controllers' point of view.

At the beginning of the thread two trajectory pointers were declared of which `newTraj` pointed at an actual trajectory object. The thread then ran while `ExitHandler::quit` was `false`. (The role of the `ExitHandler` was to ensure that threads terminated in correct order when the module was shut down.) In order to avoid dynamic memory allocation in the loop, a scheme with three statically allocated trajectory pointers was used. Incoming trajectories were written to the memory location pointed at by `newTraj`. At this point a trajectory counter (used for debugging purposes) was incremented, followed by the locking of the `m_trajRecvdMutex`. The next three lines were 'pointer juggling', making `m_recvdTraj` point at the newly received trajectory and `newTraj` at the previously received trajectory object (which was a safe location to overwrite, once a new trajectory arrived from the traffic planner).

## 7.7 South Face communication

The individual controllers wrote their output directly into actuator directives. These were given ID numbers before being queued at the south face and sent to the hardware actuating software module. The ID numbers were later used to match responses from the hardware actuating software module to their corresponding directives. (The directive-response pair was handled upon which the directive was dequeued.) To be time-wise robust against lost or late responses, the queue was implemented as a linked list (rather than vector).

## 7.8 Contingency Management

To be successful, the trajectory tracking controller module needed to handle various special situations and errors. A control status object was passed by reference to the different controllers and maintained a centralized notion of execution state. It was populated by members of the following enumeration:

```
enum followerStates {
  Running = 0,
  TrajectoryTimeout = 1 << 0,
  TrajectoryError = 1 << 1,
  EstopPause = 1 << 2,
  EstopDisable = 1 << 3,
  TransmissionPending = 1 << 4,
  TransmissionRejected = 1 << 5,
  SteeringRejected = 1 << 6,
  ThrottleRejected = 1 << 7,
  BrakeRejected = 1 << 8,
  TrajectoryEnd = 1 << 9,
  LateralMaxError = 1 << 10,
  LongitudinalMaxError = 1 << 11,
  SteeringFailure = 1 << 12,
  ThrottleFailure = 1 << 13,
  BrakeFailure = 1 << 14,
  TransmissionFailure = 1 << 15,
  ReactiveObstacleAvoidance = 1 << 16
};
```

These were all defined as powers of two, facilitating straight forward hosting of the control status in a single integer. Each member was represented by the bit of corresponding value and bit-wise masking was used to set and clear bits. This also explains why the nominal `Running` state was represented by 0.

The `LateralMaxError` and `LongitudinalMaxError` bits were deprecated. (They were originally used to request a spatial re-plan from the traffic planner, if either the lateral or speed error exceeded thresholds given in a parameter file.) The `ReactiveObstacleAvoidance` bit is not treated here, since it falls outside the context. Remaining enumeration members can be divided into trajectory- and actuator errors, given separate attention below.

#### Trajectory Errors

The `TrajectoryTimeout` bit was set when a trajectory was not received for some parametrized time (nominally 2 s). The controller responded to these timeouts by bringing the vehicle to a stop. Once a new trajectory was received, the timeout bit was cleared.

The `TrajectoryError` bit was set if a received trajectory did not adhere to the specified format. When set, a request for a new trajectory was sent to the traffic planner.

#### Actuator Errors

Actuator failures were detected by the software module receiving directives from the trajectory tracking controller. This module sent responses to all directives received from the controllers. There were two possible error responses – rejected or failed, each associated with a specific actuator. A directive was failed if the corresponding actuator suffered from a hardware failure, whereas the rejection mechanism was part of a low level contingency management scheme. For instance, throttle directives were rejected while the vehicle was changing gears (indicated by the `TransmissionPending` bit) or if there was a steering failure.

Apart from the above mentioned errors, the actuating software module was also responsible for updating the E-stop state, which was quarried and handled by the trajectory tracking controller module.

## 7.9 User Interface

To facilitate debugging, both during simulation and field tests, the trajectory tracking controller module was equipped with a partly interactive user interface consisting of several parts, handled below.

#### Sparrow Hawk

Sparrow Hawk [25] is a collection of programs and a library of C functions intended to aid in the implementation of real time controllers on PC-based data acquisition and control systems. The part of it used in Alice was a framework for implementing interactive text-based console displays. They were interactive, since keys could be bound to trigger call-back functions. It was also possible to alter the values of program variables directly from the console display.

A 'screen shot' of the Sparrow Hawk display of the trajectory tracking controller module is shown and explained in Appendix C.

### Map Viewer Visualization

The geometric nature of the lateral control algorithm, was very convenient when it came to debugging. The 'handle' described in Section 5.3 was drawn in the map viewer software, providing visual information which was easier to interpret (during run time) than the corresponding numbers in the Sparrow Hawk display.

### Parameter Files and Command Line Arguments

Gengetopt [3] is a tool for writing command line option parsing code for C programs. It was used for command line parsing at module startup as well as parsing of the controller parameter file, which could be updated from the sparrow hawk display during run time. This was very convenient, since it allowed new parameter sets to be loaded without having to restart the entire navigation stack.

### Log Files

The perhaps most important part of the user interface was the logger mentioned in Section 7.5. The following information was logged once per control cycle:

- *Timing information.*
- *Vehicle state.*
- *Actuator state.*
- *Reference trajectory information.*
- *Control errors.*
- *Timing information.*
- *Vehicle state.*
- *Actuator state.*
- *Reference trajectory information.*
- *Control errors.*
- *Controller state.*

Since file access was slow, with undeterministic duration, all log file writing was handled after the invocation of the actual controllers.

Log files were used extensively to debug and tune the controllers. They were also the basis for all material presented in Chapter 8.

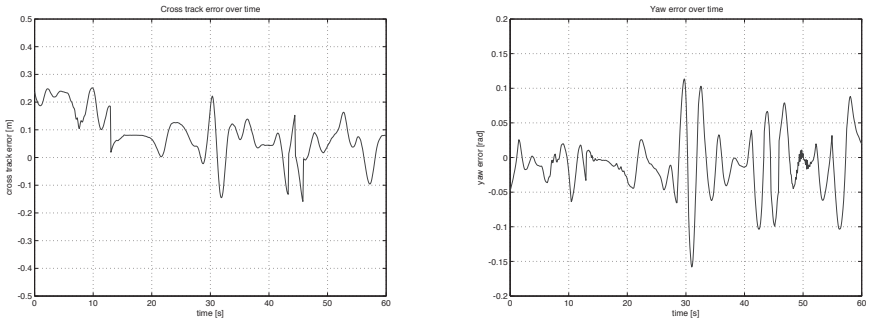
# 8. Field Results

This chapter is divided into two sections. The first, Section 8.1, shows the performance of the controllers in the field. The second, Section 8.2, explains what failed during the NQE – preventing Team Caltech from participating in the final event of the 2007 DUC.

## 8.1 Controller Performance

At the beginning of the project, the controllers were unit tested by means of static trajectories and speed profiles. Results in this section are, however, from tests where the trajectory tracking controller module was run together with the sensing and navigation software explained in Sections 3.2,3.3.

Figures 8.1(a),8.1(b) show the cross track- and yaw error, respectively, plotted during 60 s of autonomous driving. The vertical jumps seen at 13 s, 43 s and 46 s in Figure 8.1(a) were due to lateral shifts of the reference trajectories, introduced by the traffic planner. (Because of their lateral nature, these shifts affected the yaw error only marginally and are therefore not explicitly seen in Figure 8.1(b)).




(a) Cross track error $e_y$ over time.

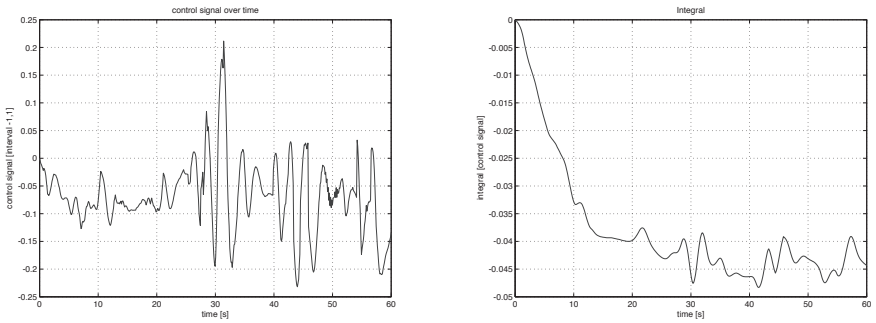
(b) Yaw error $e_\theta$ over time

**Figure 8.1** Cross track- and yaw error during 60 s of autonomous operation of the vehicle.

The control signal during the 60 s during which the data for Figures 8.1(b),8.1(a) were collected is shown in Figure 8.2(a). As seen in the figure, the signal lay within $[-0.25, 0.25]$ which is to be compared with the available range $[-1, 1]$. The integral started at 0 and converged to $-0.043$. The small value indicated that the used model was adequate, i.e. that unaccounted tyre forces etc. were negligible.

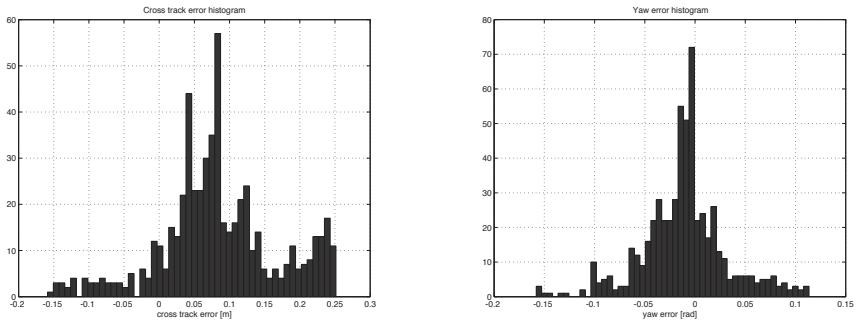
Due to degrading steering calibration, the stationary value reached by the integral changed between runs. The steering had been recalibrated shortly before the particular run, explaining why the integral converged to such a small value. Notice also the relative long time ($\sim$ 30 s) it took for the integral to converge, as discussed in Section 5.3.

Finally, Figures 8.3(a),8.3(b) show distribution histograms of the cross track- and yaw errors respectively from 10 min of continuous autonomous operation. It was difficult to draw any quantitative conclusions due to the lack of reference data obtained e.g. from the same system, with different controllers.




(a) Control signal $u_a$ (b) Integral part

**Figure 8.2** Control signal and integral from the data set used to plot Figures 8.1(a),8.1(b).




(a) Cross track error $e_\theta$ histogram (b) Yaw error $e_y$ histogram

**Figure 8.3** Histograms showing the distribution of cross track- and yaw error from 10 minutes of nominal operation.

## 8.2 The NQE – What Went Wrong

Team Caltech made it to the NQE and successfully completed one of the qualifying runs. Alice did, however, fail a merging test, which marked the end of the race for Team Caltech. The objective was to demonstrate safe mering via a left turn into traffic. (The traffic was made up by stunt driver driven vehicles). Alice was to drive up to a stop line at an intersection and come to a complete stop. The stunt cars were driving through the intersection, from right to left, with a spacing of $\sim 10$ s. Alice was then supposed to make a left turn and merge with the stunt cars in a safe manner. This is what happened:

Stop lines were defined in the RNDF. They were also sensed by stereo vision and placed in the map database of Alice. Normally, Alice stopped nicely at sensed stop lines. However, when getting close enough to a stop line, the line would eventually fall out of view of the stereo cameras. At this point it was removed from the map and replaced with the RNDF stop line. This sometimes caused a 'bunny hop' if the RNDF stop line was further ahead than the sensed one. The night before the particular NQE run, a change was made in the software, so that sensed stop lines were not removed from the map, as they disappeared out of view. The new code was tested in simulation, with play-back log data from previous tests and nothing out of the ordinary was found. However, there was a bug in the map which was previously hidden by the deletion of sensed stop lines from the map as they were approached. If the stop line perceptor returned false positives, the map would not always return the

position of the closest sensed stop line to the RNDF position, as it was supposed to.

At the NQE run the stop line perceptor returned a false stop line some meters before the real one, which was also returned. At first the map software did the right thing, but after passing the false stop line by 3.2 m, it erroneously associated the false stop line with the RNDF. (This would not have happened with the deletion of stop lines mentioned above.) Now, however, the planner logic interpreted the situation as if Alice had overshot the real stop line by 3.2 m and made it into the middle of the intersection without checking for clearance. Since this was most likely a bad place to be, the planner tried to get Alice out of intersection, but in fact drove straight into it due to the shifted map. There was a threshold, preventing this evasive behavior in situations where the stop line was only slightly overshot. However, the threshold was set to 3.0 m.

The competition coordinators saw Alice launching into the intersection and commanded an E-stop pause, preventing it from running into the oncoming traffic.

Alice was given a second try in which something similar happened. Somehow (the exact reason was never found) a point 62 m from the RNDF stop line was erroneously associated with the real stop line. This time Alice was disqualified from further competition for safety reasons.

# 9. Conclusions

A trajectory tracking software module was designed, analyzed, implemented and tested. A lot of time and effort was spent integrating it with the other modules of the system. In parallel with this, the other modules of the system were developed, integrated and tested. By the NQE all desired functionality was in place. However, there was not enough time for proper bug fixing and tuning. This became apparent at the NQE, as indicated in Section 8.2. Here follows a list of conclusions, which were realized throughout the project. Some conclusions are also found in Chapter 10.

- *Test time.* About one month prior to the NQE, there was a major code restructuring in which the main traffic planner was basically re-written. At this point a lot of time was spent testing, and it got quite hectic as new bugs were found and fixed, just to discover that they hid others. The lesson learned here, was that it took more time than one would initially imagine, to properly test and tune a complex system like Alice and that late code changes should be avoided, if possible.
- *Special cases.* The design of Alice was complicated by the handling of various special cases, which were introduced since it was hard to know what to expect at the finals. It is my belief that a more simplistic approach to the problem might well have taken Team Caltech to the finals. But once again, it was impossible to know what situations were to be encountered. For example a lot of time was spent developing code which would take Alice past road block, just to discover that there were none at neither the NQE nor the final event.
- *Team work.* Working in a big group was a challenge. Good communications were paramount to avoid misunderstandings, leading to bugs. Using the utilities, mentioned in Section 2.2, truly facilitated this work. It was also facilitated by the friendliness and motivation found in all team members.
- *CSS.* The CSS, introduced in Section 7.1, was based on a good idea. However, the implementation was rather complicated. As a result it was fully inherited by only one or two software modules. Hence, its role as a standardizing interface was somewhat lost.
- *Personal reflections.* Prior to the project, it was difficult to know what to expect. Initially, the mere amount of project code was chocking. I had never worked on a software project involving more than two persons and never written anything in C++. Fortunately, it did not take more than one or two weeks to get started. Then the entire project was one long learning experience – partly because of my limited background in software engineering. It was very stressful at times, but at the same time I cannot think of a better way to learn practical problem solving through programming. Last but not least, participating in Team Caltech has given me new friends, from around the world.

# 10. Future Work

Although the controllers did their job in tracking reference trajectories, there remained things which could potentially improve the trajectory tracking controller module and the understanding of its behavior. The areas of these possible improvements are the subjects of Section 10.1, whereas Section 10.2 deals with possible improvements of the entire system. Generally, this chapter reviews known problems, rather than suggesting their solutions.

## 10.1 Controller Module

This section is divided into three sub-sections, addressing the longitudinal control, lateral control and control related hardware, respectively.

**Longitudinal Control**

- *Position controller.* At low speeds, the non-linearity in the longitudinal actuation was not perfectly captured by the lookup table introduced in Section 5.1. (E.g. stick-slip friction, which occurred when accelerating from zero speed, was not modelled at all.) To increase smoothness of operation, especially in parking situations, a position- (rather than speed-) tracking controller, as suggested in Section 5.2 would be valuable.
- *Adaptive identification.* No explicit attention was made to changes in road composition (asphalt, dirt, etc.), winds, vehicle load or other factors affecting the validity of the lookup table shown in Figure 5.1. In general, the controlled system was robust to such changes. However, utilizing e.g. an adaptive update of the lookup table would probably have increased performance, given slow enough update times not to jeopardize corruption of the lookup table due to high frequency phenomena.

**Lateral Control**

- *Error convergence.* Apart from the numerical analysis of the lateral controller given in Chapter 6, it would be interesting to obtain a lower bound of its error convergence rate.
- *Stability margins.* The delay margin was obtained for the linearization of the system around its zero error equilibrium in Section 6.2 and verified through field experiments (not presented here). In addition to this it would be of interest to obtain measures or sensitivity to disturbances introduced at different points of the loop. (The non-static components of such disturbances were shown to be negligible in Alice, through field tests.) In addition to this, further analysis of the nonlinear region of operation would provide deeper understanding of the controller.
- *Gain schedule.* It could be worthwhile to experiment with the gain schedule of $l_2$, which was introduced in Section 6.2. A possible candidate would be e.g. a schedule resulting in a speed independent upper limit of centripetal acceleration vehicle acceleration.

- *Steer dynamics.* It was assumed that the velocity planner generated speed profiles which were feasible with respect to the slowest time constant of the steering dynamics. Further, a feed forward reference trajectory point was introduced in Section 5.3, to balance the effects of steering actuation latencies. It would be interesting, though more complicated, to analyze a more general case, where the steering dynamics are explicitly recognized as part of the controlled system.
- *Delay estimation.* The actuation delay estimate used to obtain the feed forward trajectory point was obtained offline. Introducing an estimator in the loop would make the system less sensitive to low frequency changes in the actuation delay. (High frequency changes were never a problem, since they were efficiently cancelled by the mechanical parts of the system.)

**Hardware**

- *Steering.* There was a measured delay $\tau_\phi = 400$ ms in the steering actuation. At low speeds the long delay did not pose a serious problem. However, at higher speeds ($v > 10$ ms$^{-1}$) it sometimes triggered an oscillative behavior in the lateral controller when the reference trajectory was suddenly shifted laterally by the traffic planner. This was a difficult problem to approach. Simply reducing controller gain was not a feasible solution, since it would degrade tracking performance to an unacceptable level before eliminating the swerving. Instead, attempts were made to identify the trajectory shifts and temporarily switch to a less aggressive control strategy. The problem was eventually eliminated by improvements in the traffic planner, which prevented it from generating suddenly shifting reference trajectories.
- *Braking.* The maximal deceleration when applying longitudinal control signal $u_a = -1.0$ was $-2.7$ ms$^{-2}$ in Alice. From the point of view of the longitudinal controller, there was nothing to do about this low value. However, the resulting long stop distance posed a serious problem in traffic planning. With speeds of $v > 10$ ms$^{-1}$, larger decelerations would be required in order to safely stop the vehicle in case of suddenly appearing obstacles.

## 10.2 System

The trajectory tracking controller module was only a small part of a complex system.

- *Vehicle state estimation.* Throughout the development process, problems were experienced with the Applanix vehicle state estimator described in Section 3.1. Occasionally it introduced large ($> 1$ m) state drifts as the vehicle resumed driving after long ($> 30$ s) stops. Unfortunately, the problem was never resolved, since the team did not have access to the software of the Applanix – it was merely a black box, providing state data. Instead a workaround solution utilizing drifting coordinate frames was introduced.
- *Unit testing.* A large amount of system test time was inefficiently spent, since it was difficult to unit test some of the individual modules before integrating them into the system. A proper framework for unit testing would therefore have facilitated the development process significantly.
- *Sensor coverage.* Although Alice was equipped with a seemingly large number of sensors, sensor coverage proved to be inadequate in e.g. intersections. This was worsened by the fact that the perception software sometimes confused static- and moving obstacles.

# A. Project Time Line

The nominal time line of Team Caltech's development process for the 2007 DUC was divided into 5 spirals, each defined by a list of goals. The spirals and corresponding goals are given below. A brief discussion of the time line was given in Section 2.3.

***Spiral 0 – Planning and Preparation (13 Jun - 8 Jul)***

1. Finalize plan for the summer (goals, time line, sub-teams).
2. Update architecture (navigation system, coding standards).
3. Update software infrastructure (unit test structure, common startup scripts, YaM improvements).
4. Update hardware infrastructure (remount sensors, cabin reorganization, finalize power design, install new computers).

***Spiral 1 – Navigation and Traffic (9 Jul - 5 Aug)***

1. Safe and robust driving in all navigation- and basic traffic scenarios .
2. Working versions of all modules.
3. Hardware system finalized.

***Spiral 2 – Advances Traffic (6 Aug - 2 Sep)***

1. Safe, robust and persistent driving in all navigation- and traffic scenarios.
2. System functionality finalized.

***Spiral 3 – Optimization (4 Sep - 30 Sep)***

1. Race-ready operations and capability of autonomously driving 60 miles in less than 6 hours.
2. Code finalized.

***Spiral 4 – NQE and Race Preparations (1 Oct - 21 Oct)***

1. Mistake-free NQE and race procedures.

# B. Team Members

The chart below lists the members of Team Caltech, who were involved in the preparations for the 2007 DUC. (Persons listed below the horizontal lines participated during the 2006-07 academic year, but were not involved in the development covered in Appendix A.)

| *Team leader:* Richard Murray, *Team co-leader* Joel Burdick | | | |
|---|---|---|---|
| *Vehicle Team* | *Navigation Team* | *Sensing Team* | *Systems Team* |
| Dominic Rizzo (coord) | Noel duToit (coord) | Sam Pfister (coord) | Nok Wongpiromsarn (coord) |
| Tony & Sandie Fender | Vanessa Carson | Mohamed Aly | Josh Doubleday |
| Rob Grogan | Stefano di Cairano | Andrew Howard | Jessica Gonzalez |
| Noele Norris | Sven Gowal | Laura Lindzey | Phillip Ho |
| Josh Oreman | Magnus Linderoth | Jeremy Ma | Robbie Paolini |
| Jimmy Paulos | Christian Looman | Humberto Pereira | Rich Petras |
| Glenn Wagner | Mark Milam | Christopher Rasmussen | Chris Schantz |
| Daniel Alvarez | Kenny Oslund | Henrik Sandberg | Luke Durrant |
| William David Carrillo | Jeremy Schwartz | Tamas Szalay | Julia Braman |
| Arthur Chang | Kristian Soltesz | Daniele Tamino | Edward Chen |
| Iain Cranston | Francisco Zabala | Pete Trautman | Steve Chien |
| Matthew Feldman | Lars Cremean | Joe McDonnell | Jay Conrod |
| Nicholas Fette | Tom Duong | Brandt Belson | Scott Goodfriend |
| Ken Fisher | Luke Durant | Philipp Boettcher | Mitch Ingham |
| Nikhil Jain | Melvin Flores | Justin McAllister | Michael Kaye |
| Michael Kaye | Steven Gray | Miles Robinson | aditya Khosla |
| Daniel Talancon | Russell Newman | David Trotz | Bob Rasmussen |
| Albert Wu | Brent Goldman | Yi Wang | Chess Steton |
| | Ghyrn Loveness | Johnny Zhang | Sashko Stubailo |
| | Jerry He | | |

# C. Sparrow Hawk Display

The sparrow Hawk display was divided into three columns – one per controller. A forth column (the rightmost) was used to display status messages (not shown here) and trajectory information. Further, the display was vertically divided into state-, timing- and error sections. The bottom part was dedicated to reactive obstacle avoidance (not treated here). A 'screen shot' of the Sparrow Hawk display during nominal operation is shown below.

```
SNKEY: 0                                Hit 'l' to toggle visualization in mapviewer
--------------------+--------------------+------------------+-------------------------
Lateral (steering)  |Longitudinal (speed) |Transmission      |Status
--------------------+--------------------+------------------+-------------------------
error:   0.174      |error:  %0.375       |trajDir:  1       | Traj ID:    784
I-part:  %la_I      |I-part: %lo_I        |Cur gear: 1       | Traj age:   0.27
phi:     %la_phi    |vRef:   %lo_vR       |Pending:  0       | astate age: 0.12
phi FF:  %la_pFF    |vel:    %lo_vel      |                  |
l1 GS:   %la_l1G    |a  :    %lo_a        |                  |
l2 GS:   %la_l2G    |aFF:    %lo_aFF      |                  |
                    |aPitch: %lo_aP       |                  |
u:       %la_u      |u:      %lo_u        |                  |
%la_replan          |%lo_replan           |                  |<-Replan
--------------------+--------------------+------------------+
1007                |1004                 |%dir_trans        |<-Directives
1005                |1002                 |%resp_trans       |<-Responses
True                |True                 |False             |<-Sending
0.274               |0.283                |                  |<-Loop age
0.182               |0.194                |                  |<-State age
--------------------+--------------------+------------------+
                    |Gas:     |Brake:     |                  |
                    |         |           |                  |<-Rejected
                    |         |           |                  |<-Failed
--------------------+--------------------+------------------+
Trig: %roa_ot   Dist: %roa_cd  Dec: %roa_dt  Del: %roa_dd    |<- ROA [m]
----------------------+-------------------------------------+
%roa_m %roa_z %roa_or  |MASTER  DELY  OBSTACLE    %roa_ben|<- ROA state
----------------------+-------------------------------------+
                                              |           |
2 updatess of parameters from file:           +-----------+
/home/users/team/alice-hardware-01/etc/follower/paramfile


QUIT        update
```

The display also showed from which file the current controller parameters were loaded. It enabled updates of these as well as the ability to toggle visualization of the lateral control algorithm in the graphical map viewer software.

# References


[1] The Bugzilla web page. Visited October 2007.
http://www.bugzilla.org. 2.2

[2] The Doxygen web page. Visited November 2007.
http://www.stack.nl/~dimitri/doxygen. 2.2

[3] The Gengetopt web page. Visited November 2007.
http://www.gnu.org/software/gengetopt/gengetopt.html. 7.9

[4] The Red Racing Team web page. Visited October 2007.
http://www.redteamracing.org. 1.2

[5] The Stanford Racing Team web page. Visited October 2007.
http://cs.stanford.edu/group/roadrunner. 1.2

[6] The Subversion web page. Vistied October 2007.
http://www.subversion.tigris.org. 2.2

[7] The Tartan Racing Team web page. Visited November 2007.
http://www.tartanracing.org. 1.2

[8] The Team Caltech Wiki web page. Visited October 2007.
http://gc.caltech.edu/wiki07. 2.2

[9] The Virginia Tech Team web page. Visited November 2007.
http://www.me.vt.edu/urbanchallenge. 1.2

[10] The YaM Wiki web page. Visited October 2007.
http://dartslab.jpl.nasa.gov/yam.
The site requires user name and password authentication. 2.2

[11] R. H. Byrne, C. T. Abdallah, and P. Dorato. Experimental Results in Robust Lateral Control of Highway Vehicles. *IEEE Control Systems Magazine*, pages 70–76, April 1998. 5.3

[12] A. corp. *POS LV V4 Installation and Operation Guide*, 2006. 3.1

[13] L. B. Cremean, T. B. Foote, J. H. G. illula, G. H. Hines, D. Kogan, K. L. Kriechbaum, J. C. Lamb, J. Leibs, L. Lindzey, C. E. Rasmussen, A. D. Stewart, J. W. Burdick, and R. M. Murray. Alice: An Information-Rich Autonomous Vehicle for High-Speed Desert Navigation. *Journal of Field Robotics*, Issue 9, Part 2, September 2006. 2.1, 3.2

[14] DARPA. 2004 Grand Challenge web page. Visited October 2007.
http://www.darpa.mil/grandchallenge04. 1.2

[15] DARPA. 2007 Urban Challenge web page. Visited August 2007.
http://www.darpa.mil/grandchallenge/index.asp. 1

[16] DARPA. DARPA 2005 Urban Challenge web page. Visited October 2007.
http://www.darpa.mil/grandchallenge05. 1.2

[17] DARPA. *Urban Challenge Route Network Definition File (RNDF) and Mission Data File (MDF) Formats*, March 2007.
Available at: http://www.darpa.mil/grandchallenge. 6

[18] DARPA. *Urban Challenge Rules*, July 2007.
Available at: http://www.darpa.mil/grandchallenge. 6

[19] H. Fritz. Longitudinal and lateral control of heavy duty trucks for automated vehicle following in mixed traffic: experimental results from the CHAUFFEUR project. In *Proceedings of the 1999 IEEE International Conference on Control Applications*, pages 1348 – 1352. IEEE, August 1999. 1.1

[20] G. Indiveri. Kinematic Time-invariant Control of a 2D Nonholonomic Vehicle. In *Proceedigs of the 1999 IEEE Conference on Decision and Control*, pages 2112–2117. IEEE, December 1999. 5.3

[21] A. Jain and J. Biesiadecki. YAM- A Framework for Rapid Software Development. In *Proceedings of the second IEEE International Conference on Space Mission Challenges for Information Technology*. IEEE, July 2006. 2.2

[22] A. Kampfer and E. Pucher. Principles of systems that enable autonomous driving of vehicles. In *Proceedings of the 2005 IEEE Conference on Intelligent Transportation Systems*, pages 930–935. IEEE, September 2005. 1.1

[23] Y. Kanayama, Y. Kimura, F. Miyazaki, and T. Noguchi. A Stable Tracking Control Method for an Autonomous Mobile Robot. In *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, pages 384–389. IEEE, May 1990. 5.3

[24] A. Kelly. A Feedforward Control Approach to the Local Navigation Problem for Autonomous Vehicles. Technical report, The Robotics Institute, Carnegie Mellon University, Pittsburgh, USA, May 1994. 5.3

[25] R. M. Murray. *Sparrow Primer*, July 1995. 7.9

[26] R. M. Murray *et al.* Team Caltech Technical Paper – DARPA Grand Challenge. Technical report, Department of Control and Dynamic Systems, California Institute of Technology, Pasadena, USA, February 2004. 1.2

[27] R. M. Murray *et al.* DARPA Technical Paper: Team Caltech. Technical report, Department of Control and Dynamic Systems, California Institute of Technology, Pasadena, USA, August 2005. 1.2

[28] R. M. Murray *et al.* Sensing, Navigation and Reasoning Technologies for the DARPA Urban Challenge. Technical report, Department of Control and Dynamic Systems, California Institute of Technology, Pasadena, USA, April 2007. 3.2

[29] M. Netto, J.-M. Blosseville, B. Lusetti, and S. Mammar. A new robust control system with optimized use of the lane detection data for vehicle full lateral control under strong curvatures. In *Proceedings of the IEEE Intelligent Transportation Systems Conference*, pages 1382–1387. IEEE, September 2006. 5.3

[30] S. L. Tan and J. Gu. Investigation of Trajectory Tracking Control Algorithms for Autonomous Mobile Platforms: Theory and Simulation. In *Proceedings of the IEEE International Conference on Mechatronics & Automation*, pages 934–939. IEEE, July 2005. 5.3

[31] Team Caltech. Team Caltech web page. Visited August 2007.
http://gc.caltech.edu/public/Main_Page. 1.2

[32] W. Weiguo and W. Yuejuan. A Novel Global Tracking Control Method for Mobile Robots. In *Proceedings of the 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 623–628. IEEE, October 1999. 5.3