JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 4, December 2007

Empirical Evaluation of an Autonomous Vehicle in an Urban Environment

Ben Upcroft*†, Alexei Makarenko† , and Michael Moser†
University of Sydney, NSW, 2006, Australia

Alen Alempijevic‡ and Ashod Donikian‡
University of Technology Sydney, NSW, 2007, Australia

and
Will Uther§ and Robert Fitch§
National ICT Australia, Kensington, NSW, 2052, Australia

DOI: 10.2514/1.32839

Operation in urban environments creates unique challenges for research in autonomous ground vehicles. Due to the presence of tall trees and buildings in close proximity to traversable areas, GPS outage is likely to be frequent and physical hazards pose real threats to autonomous systems. In this paper, we describe a novel autonomous platform developed by the Sydney-Berkeley Driving Team for entry into the 2007 DARPA Urban Challenge competition. We report empirical results analyzing the performance of the vehicle while navigating a 560-meter test loop multiple times in an actual urban setting with severe GPS outage. We show that our system is robust against failure of global position estimates and can reliably traverse standard two-lane road networks using vision for localization.

I. Introduction

THE DARPA Grand Challenge (DGC) competitions have a history of pushing boundaries of applied research in robotics and autonomous systems. The 2007 competition is unique in that it is set in an urban environment, posing new challenges in system robustness and reliability. The Sydney-Berkeley Driving Team (SBDT) entered the 2007 DARPA Urban Challenge with an interest in attacking some of these challenges. This paper describes our entry: a modified Toyota RAV4 (Fig. 1).

The Sydney-Berkeley Driving Team is a collaboration between the University of Sydney, the University of California at Berkeley, the University of Technology Sydney, and National ICT Australia (NICTA). Our main goal was to build a platform that could operate in an *actual* urban environment (Fig. 2). Specifically, real urban environments involve: 1) poor Global Positioning System coverage (GPS outage), 2) physical hazards such as buildings, trees, or other obstacles in close proximity to roads, and 3) difficulties for vision systems due to changing lighting conditions. In addition, we were interested in addressing these challenges using relatively inexpensive and commonly available hardware and sensors.

Received 15 June 2007; revision received 15 October 2007; accepted for publication 15 October 2007.

* Contact Author: b.upcroft@cas.edu.au
† Australian Centre for Field Robotics, The Rose St. Building J04
‡ Mechatronic and Systems Intelligence Group
§ Neville Roach Laboratory, 223 Anzac Pde

Fig. 1 The SBDT Toyota RAV4 autonomous vehicle.

Our approach was to develop a robust system that could drive safely, minimizing unrecoverable errors. The overall strategy is to use local sensing to compensate for errors in global position estimation, and to use a hierarchical planning system to make intelligent high-level decisions.

Our system design incorporated many existing robotics techniques, including probabilistic state estimation, terrain modeling, sensor fusion, sample-based planning through the terrain map, and feedback control at multiple levels of abstraction. A novel aspect of the design compared with many robots from the previous DARPA Grand Challenges was vision-based localization using lane markings (Fig. 3). Our test facility was particularly complex with GPS satellite acquisition denied over large sections (100s of meters) of the course. For this reason, a robust, local navigation solution through the combination of vision and dead reckoning was required.

One of the design requirements for the RAV4 was the use of a minimal set of relatively inexpensive sensors for traversal of an urban environment where global localization information was not always available. Using a single camera, a set of three 2D laser range finders, a GPS receiver, and an Inertial Measurement Unit (IMU), the RAV4 was successfully tested at speeds of up to 6 ms^{-1} (13 mph).

We conducted experiments in an urban field site near Berkeley, California. The site had severe GPS outage due to tall trees and buildings, and also narrow lanes and frequent obstacles. Our performance in the DGC site visit was hindered by the difficulty of the test course. However, in spite of this, we took the opportunity to make

(a) (b)

Fig. 2 (a) Our test facility – an *actual* urban environment. (b) The test facility as viewed from the vehicle.

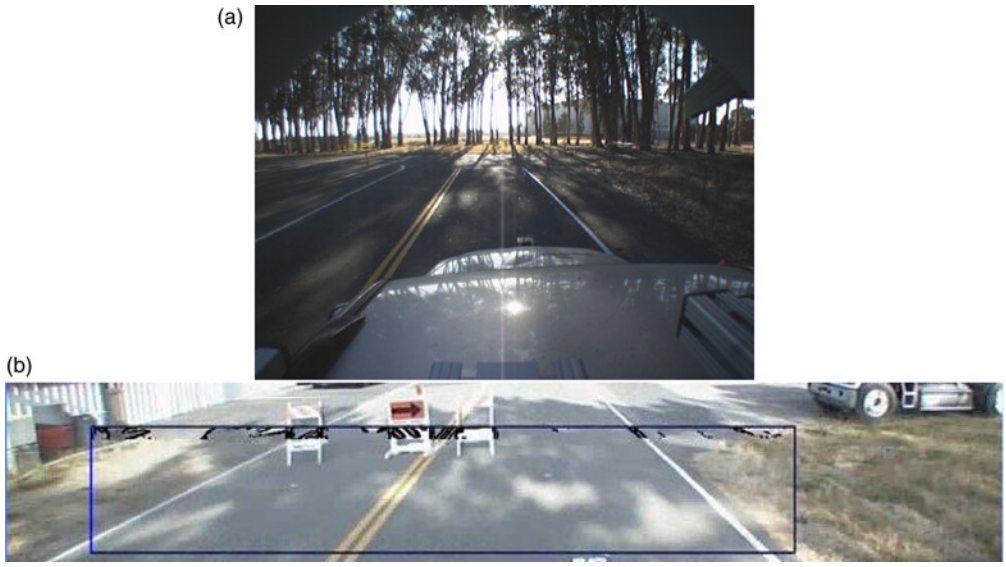

Fig. 3 Lane markings as viewed from the vehicle: (a) lanes in sun-dappled lighting, (b) lanes with obstacles.

improvements and continue testing in this challenging environment, enabling us to conduct a significant empirical evaluation of our system. The results we report here show that the vehicle reliably navigates around an urban course over 0.5 km in length. We were also able to perform high-level behaviors such as overtaking parked vehicles and proper queuing behavior approaching an intersection. We conclude that our vehicle is a robust platform for future research in autonomous vehicles, including future DGC competitions.

The paper is organized as follows. Sec. II presents the hardware design of the vehicle. In Sec. III we describe the overall software system architecture, including algorithms and implementation. We report results from extensive field testing in Sec. IV, and conclude with a discussion in Sec. V. We discuss related work throughout the text as is relevant. For collected related work we refer the reader to the journal special issues on previous grand challenges.[1,2]

II. The Vehicle Hardware

A stock 2006 Toyota RAV4 was used as our base vehicle. The lowest level of the control system consists of the vehicle actuation. This system was custom-designed based on commercial systems while taking advantage of the existing drive-by-wire capabilities of the RAV4. Three components are critical to the control and operation of the vehicle: throttle, brake, and steering.

For the modifications to each of these components, reliability, actuation speed, and control accuracy were considered. In addition it was clear from the experiences in the previous DARPA Grand Challenge that rigorous testing was vital for success of the project. Finally, in case of electrical, mechanical, or software failure, an emergency stop over-ride automatically shuts down all autonomous actuation and brings the vehicle to a quick stop without the need for external commands. For autonomous testing, operations, and logistics in general, the vehicle is kept in a state which facilitates switching between robotic and human control without any need for mechanical alterations. Each actuation subsystem is controlled through independent switches accessible to a human observer sitting behind the front passenger seat. This allowed a great amount of flexibility for testing since some functionality of the system could be controlled by a human driver while other parts were completely autonomous. In addition, modifying the vehicle for normal driving and returning it to a state that is street legal can be performed within minutes.

A. Actuation

1. Steering

The RAV4 is equipped with electric power steering. In normal operation a sensor measures the torque applied to the steering column by the human driver. The sensor outputs an analog signal which the power-steering electronic

control unit (ECU) interprets as a torque. The ECU then drives an assist motor which applies an assisting torque. In autonomous operation, the torque sensor signal is replaced by a simulated signal provided by an analog output card (Advantech PCI-1721) installed on the QNX onboard control computer. Care was taken to ensure that the simulated signal was close to the expected signal so that it was accepted as genuine and failsafe states were rarely triggered.

This modification allowed the vehicle's built-in safety features (stability control, damping of too aggressive inputs) to remain in operation. It also allowed the car to be returned to the original (street legal) configuration by disconnecting the custom control and connecting the original torque sensor.

2. Throttle

Throttle control in the RAV4 is achieved with an analog position sensor located in the accelerator pedal. It provides a signal to the throttle in a similar manner to that of the torque sensor for steering. As for the steering, the connector to the sensor is replaced with a custom adapter for control from our analog output card.

3. Brakes

Unlike the previous actuators, the brake does not have an existing electronic interface and thus required retrofitted actuation. In the previous Grand Challenge there were three main choices for brake actuation: linear electric actuators (used in the majority of vehicles for DARPA Grand Challenge, 2005), (electro-) hydraulic actuators, and pneumatic actuators (a sizable minority, 2005). Linear electric actuators allow good position control. However, their major drawbacks are relatively slow speed and the lack of force control. Although hydraulic systems provide good position and force control, we dismissed this option in the early stages since there was little relevant experience in the use of these systems within the team.

We chose a pneumatic system as it is fast, allows good force control, and we had experience within the team using these systems. Note that one drawback with pneumatic actuators is that they are limited in position control. We also desired the ability for parallel brake actuation by a robotic and human driver as described in several designs for the 2005 Grand Challenge.

To avoid space constraints in the driver's seat, the vehicle was converted to dual brake controls (as in driver training vehicles) and the front passenger seat removed. This conversion was performed by a certified mechanic ensuring the vehicle remained street legal. The brake actuator (Fig. 4) could then be mounted to the existing strong points designed for the seat and attached to the dual brake guide cable (which would normally connect to an additional brake pedal). Note that for manual driving, the guide cable is mechanically detached from the actuator. The pneumatic cylinder is controlled using the same analog output card as the other actuators. The cylinder pulls the guide cable longitudinally to apply the brakes.

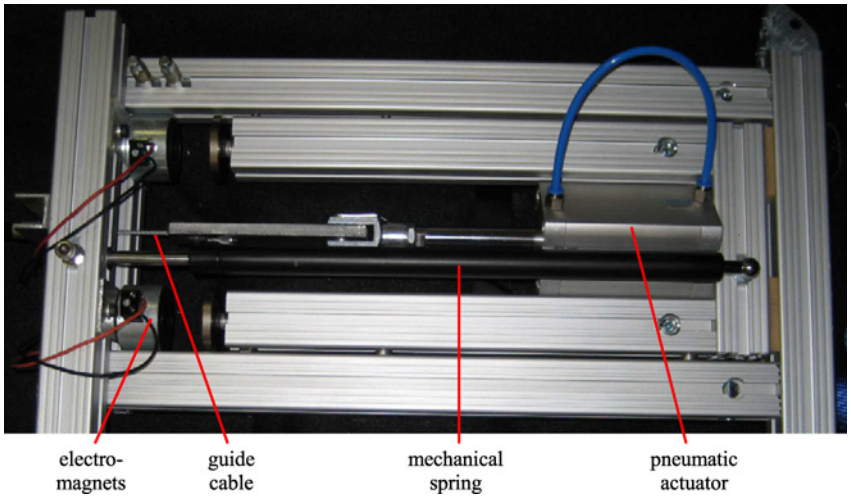

Fig. 4 Pneumatic brake actuator and failsafe system.

To ensure a failsafe system, we adopted the approach of the Austin Robot Technology Team from the 2005 DARPA Grand Challenge.[3] The U-shaped assembly, termed the *sled* (constructed from Maytec rails), around the cylinder can move relative to the rest of the frame. The system is shown in failsafe mode (i.e. maximum brake applied) with the mechanical spring pushing the sled to the right and hence applying force on the brake. To reset the system from the failsafe state, the sled is manually pushed forward, with assistance from the pneumatic actuator, closing the gap between the sled and the electromagnets. If power is supplied to the electromagnets, they hold the sled in the left position with the brakes released, giving the pneumatic actuator full control. This setup is failsafe in the sense that loss of power will result in the electromagnets releasing the sled and the mechanical spring applying steady braking force.

B. E-Stop

The primary concern of the e-stop system is the safety not only of the team members but also of the general public, with an additional concern of ensuring that damage to property and equipment does not occur. Furthermore, it was required that the vehicle be maintained in a manner in which it can be switched back to a street legal condition. Thus, the e-stop system implementation and control of the actuators should never be allowed to interfere with normal driving on public roads. To guarantee zero interference during normal driving the e-stop system and actuator control circuitry are physically disconnected from the vehicle controls ensuring that the two systems are isolated. The current wireless e-stop (Hetronic ERGO V4, operation frequency = 434 MHz), is designed to be easily interchangeable with the DARPA-provided e-stop via a single connector.

To maintain simplicity, our implementation of the e-stop system is limited to the three e-stop modes as specified by the DARPA guidelines: *disable*, *pause* and *run*.[4] For safety and reliability it was decided to implement the e-stop modes and their transition completely in hardware. This prevents potential software bugs from resulting in an unknown condition or state of operation.

The *disable* mode was implemented by removing power from the electromagnets. *Pause* mode supplied a hardware-tunable input to the pneumatic actuator, bringing the vehicle to a controlled stop. *Run* mode connected the pneumatic actuator to the analogue output from our computer.

C. Actuator Controllers

Since the trajectory planning is handled at the motion planning level, our low-level control interface consists primarily of vehicle speed (throttle and brake) and steering angle, both executed using Proportional-Integral-Derivative (PID) feedback loops. The controllers also consider secondary state information (e.g. which gear is currently active, car error states, etc.).

1. State Information

Using the state information provided by the vehicle's Controller Area Network (CAN) Bus and the Inertial Navigation System, we were able to accurately characterize the car's parameters. Most importantly, actuator positions could be mapped to actual states such as acceleration/deceleration, speed, and steering angle for precise low-level control.

We are using an optoisolated CAN-controller card by PEAK System (PCAN-PCI) to interface with the vehicle's CAN-Bus. This card can be used under the real-time operating system QNX, using a third-party driver by BitCtrl Systems. With this setup we achieve high data rates (500 kb/s for the vehicle's CAN-bus), high frequency (up to 80 Hz, depending on message priority), and excellent timing accuracy.

2. Steering Controller

Steering was the simplest controller and just required a PID feedback loop which corrected the error in steering angle. The built-in power steering motor provides the capability to turn the wheels from lock to lock in 1.2 seconds. However, the PID gains that we are currently using have been tuned for smooth operation and take approximately 2.0 seconds.

3. Speed Controller

The speed controller consists of two PID controllers (brake and accelerator) and an arbiter to choose the controller for a given situation. The average speed of the rear wheels (calculated from the wheel encoder information on the

vehicle's CAN-Bus) provide the feedback with corrections at a frequency of 80 Hz. The arbiter allows only one controller to run at the same time; braking and accelerating cannot occur simultaneously. It also considers the vehicle state, so that the maximum RPM cannot be exceeded and the throttle can only be applied while in gear.

Control of the brake was more difficult than the throttle due to static friction in the mechanical system. As a result, brake force was not proportional to the applied actuator pressure. Instead, the effective braking force would remain relatively constant and at irregular intervals jump to match the actuator force. Further testing showed that small variations in actuator force generally resulted in unpredictable and jumpy responses in effective braking force. However, step inputs of varying size resulted in accurate proportional brake force response.

To solve the static friction problems in the control system, we quantize the output of the brake controller. Whenever the output of the controller is commanded to change, the actuator is set to zero force for 60 milliseconds and then returns to the quantized set point. This ensures the effective brake force matches the controller's output by transforming a small correction into a large one and eliminating the effects of static friction.

D. Computing and Networking

The onboard computer system currently has four hosts (not counting diagnostic laptops that are often connected to the system). The onboard computers use two operating systems: Linux (Kubuntu) and QNX. The computing hardware consists of off-the-shelf rack-mounted PCs with Intel dual-core processors. The hosts are connected with a standard 1-Gigabit Ethernet hub. Latency performance tests with this setup are shown in Fig. 5.

Some, but not all, parts of our system require real-time features. For example, there is a strong need for accurate timestamping of sensor data. A vehicle in the competition moves at speeds of up to 30 mph (48 kph) and small

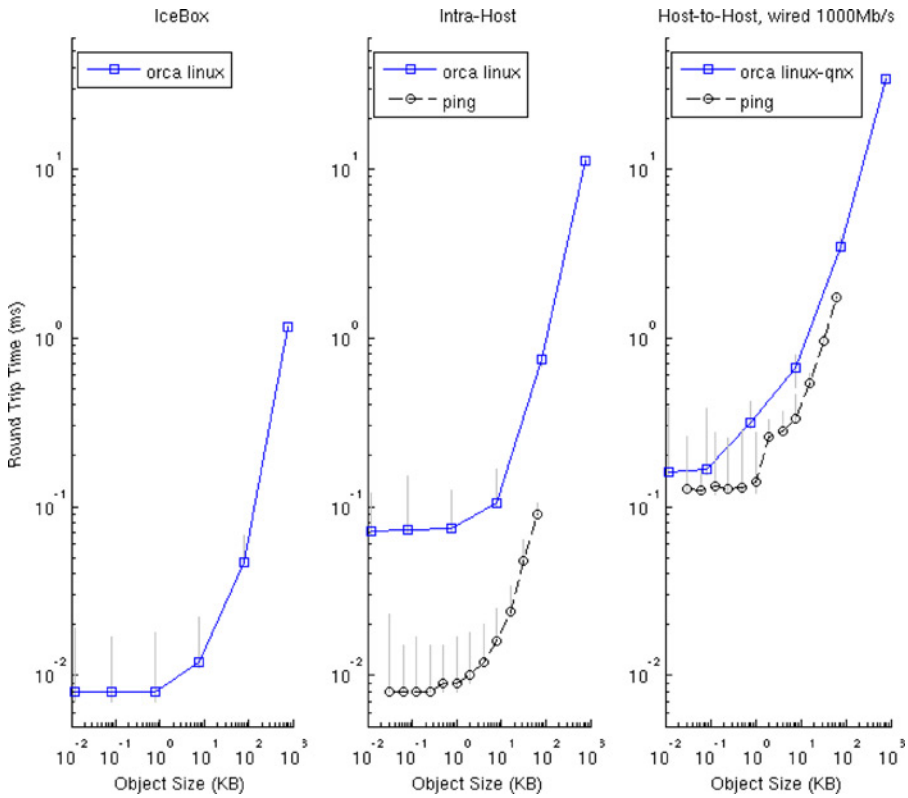

Fig. 5 The tests we perform reflect the typical usage of our components: frequent transfer of small to medium size messages. Here we provide some numbers on round trip time (RTT) for sending messages of various sizes. The plots above show average RTT for sending 100 objects with an interval of 0.25 second. Vertical bars show minimum and maximum values. Ping results are shown for comparison.

Fig. 6 The sensor suite: SICK lasers, INS, and camera.

sporadic delays in the standard Linux kernel can have significant negative impact, particularly in navigation. To address this issue, we use a dedicated host running the real-time QNX Neutrino operating system and one with the preemptive Linux kernel patch for all interactions with sensor and actuator hardware.

E. Sensors

Perception of the environment and the vehicle state is performed by onboard sensors. Most of the vehicle sensors have been mounted on the custom built roof rack (Maytec rails) shown in Fig. 6.

Visibility of the terrain is greatest on the roof and obstruction of signals received by the GPS is at a minimum in this position. Fusion of GPS and output from an Inertial Measurement Unit (IMU) provides the vehicle navigation solution. The IMU is mounted to the chassis above the transaxle. Currently, perception of the environment is performed using two SICK LMS291 lasers and a single camera (Hitachi HV-F31F, 3CCD progressive scan). One laser is pointed forwards at a pitch of 7°. This laser records the cross-section of the terrain at a range of 16 m in front of the vehicle. The second laser is mounted vertically and is used for estimating the relative pitch of the road out to approximately 25 m depending on the reflectivity of the road. The lasers are connected to a Bluestorm/Express PCI-Express 4-port serial card and communicate using RS-422 in order to run at a 500 k baudrate. The clock onboard the serial card was replaced with a clock that could provide this nonstandard speed.

The camera is used for detection of lane markings out to approximately 40 m limited by resolution of the camera. It is connected to the computers via FireWire.

Estimating the vehicle state accurately is a key requirement for sensible path planning and representation of obstacles. Errors in the pose estimate can result in poor terrain maps with phantom obstacles leading to seemingly poor control decisions. Thus it is important that the navigation solution, at least at the local level, is smooth and locally accurate. Navigation is the central sensor system (Novatel ProPak-V3 GPS receiver combined with a Honeywell HG-1700). We use OmniSTAR HP global GPS corrections. All other sensors are registered with the navigation subsystem and fused using this information.

III. System Architecture

The Urban Grand Challenge not only poses the problem of autonomy at the hardware level but also at abstract levels such as perception and decision making usually only performed by human operators. The SBDT has chosen a

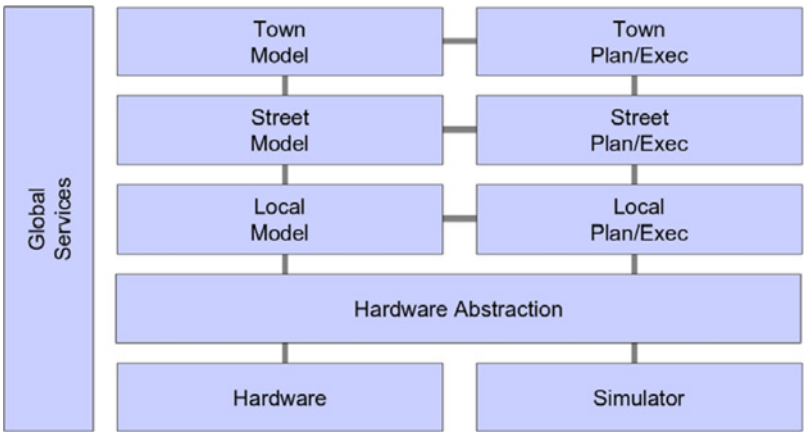

Fig. 7 The SBDT system architecture uses a classic hierarchical approach.

classic N-tier architecture[5] to address the large range of abstractions required in this competition (Fig. 7). The tiers, separated by the level of abstraction in control and perception, include modeling, planning, and execution elements.

The N-tier architecture encodes the following separation of concerns. The task of driving from point A to point B is first considered on the global level. Given the town map a trip plan is generated in terms of GPS-referenced waypoints. Travel between waypoints can only be performed by following streets (ignoring parking lots for the moment). Perception, modeling, and planning motion along a street segment in the presence of traffic and complying with traffic rules is handled by the street level of the architecture.

Local environment modeling and action planning guarantees physical safety of the vehicle as it moves towards its goal. Information about the local environment and the vehicle itself is extracted from onboard sensors. The extent of modeling and planning at this level is limited by the sensor range and a finite (short) history of past observations. Some of the representative subtasks at each of these levels include tracking the vehicle location relative to the map: identifying street boundaries, buildings and other cars, controlling vehicle motion, reasoning about traffic rules, and many others.

A. Software Architecture

Building reliable robotic hardware is a difficult task in itself, but the major challenge of this competition is in software. The main difficulties arise from the task's complexity and its dynamic real-time nature. Other contributing factors include the distributed and cross-platform computing environment, the large number of software contributors, and the need to reuse existing code.

Our software is built using a Component-Based Software Engineering (CBSE) approach. This offers modularity, software reuse, and flexibility in deployment, all of which are necessary to address the problems listed above. Applied to a robotic application, CBSE means that algorithms for perception and navigation are mapped to a set of components. Components run asynchronously and exchange information through communication.

We use ZeroC's Ice middleware[a] extensively in our system for component interface definition, inter-component communication, component deployment, location, activation services, etc. We also use Orca,[b] an open source project that customizes Ice to robotic applications and provides an online repository of reusable components.

The total number of components that comprise our current system is less than twenty. However, this is expected to increase as the complexity and number of algorithms increase (for reference, the winner of the 2005 DARPA Grand Challenge had approximately thirty.) The software is written by about a dozen people from four organizations (this number is higher if we count the authors of the existing components used directly in our system).

[a] http://zeroc.com/
[b] http://orca-robotics.sourceforge.net/

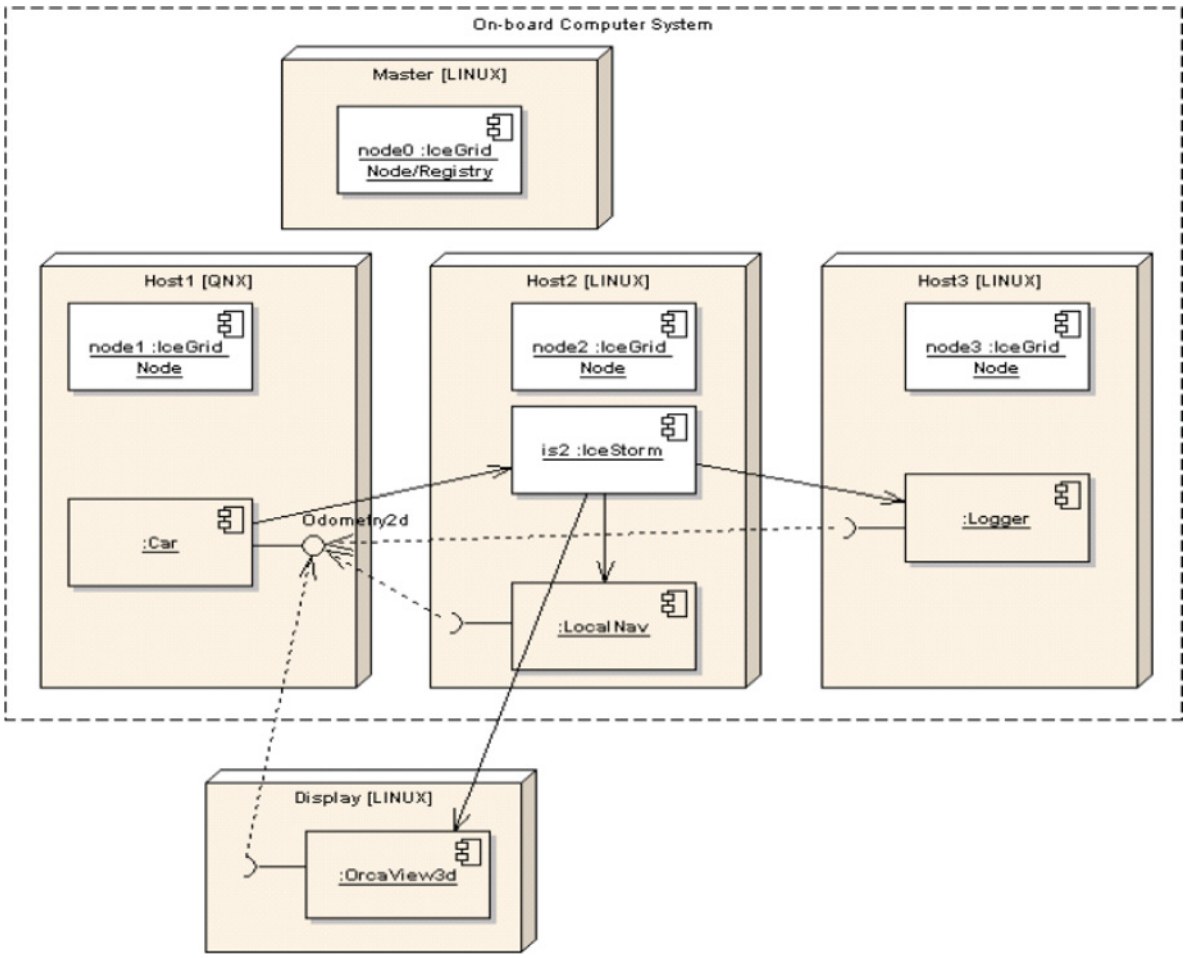

Fig. 8 Deployment diagram illustrating how the onboard computer system is configured.

Fig. 8 illustrates the current deployment strategy. The master host executes an instance of an IceGrid node with a collocated IceGrid registry. Every other host runs an IceGrid node. One host runs QNX Neutrino, the remaining hosts run Kubuntu Linux. The QNX host executes low-level actuator and sensor components (our own partial, unsupported port of Ice to QNX is available through the ZeroC developer forums). For brevity, not all system components (e.g. lasers) are shown. Some components that we use in the vehicle are currently publicly available, such as the laser range finder, the inertial navigation system, and others; due to the competitive nature of this project, other components, such as the car component that drives the vehicle are not publicly available. However, we intend to release most of these components through the Orca project in the future. Using Ice's tools we can manage the system from a centralized XML file, and distribute the system's components to computational nodes.

B. Levels of Abstraction

The levels of abstraction are reflected in the system architecture (Fig. 7). We use a layered approach to planning and control. Our approach consists of a town planner, a street-level planner, and a local motion planner. This layering achieves a number of benefits. First, it is more efficient to plan in a layered manner. We only plan in high detail through a region about which we have information. Also, it is an important design principle of the team's entry that driving is performed at a local level. We do not need to know where we are in a global space to retain safe control of the vehicle. Finally, although both the town and motion planners are in metric spaces, the layering allows these metric spaces to be decoupled. This enables us to use two navigation solutions: one smooth, but perhaps not globally

accurate (from the IMU and camera), and another one globally accurate, but usually not smooth (from the GPS unit). The local obstacle detection and tracking uses the smooth navigation solution, and the global positioning uses the globally accurate positioning. We describe the three layers of our planning system (as shown in Fig. 7) in this section. Perception and planning will be considered together for each level.

C. Town Level

The route planner reads in the Route Network Definition File (RNDF) and Mission Definition File (MDF) supplied to the team, and uses them to plan a high-level path that fulfills the mission defined in the MDF. On startup, the RNDF is processed to produce a connected graph (Fig. 9). While driving, this graph is searched each second to discover a route to the next checkpoint, and then the route is processed to determine where the vehicle should next stop or exit the current street. That next exit or stopping point is passed to the street-level planner, along with any available information regarding the current road segment such as speed limits, estimated Euclidean distance to the exit point, and road curvature. The algorithm is summarized in pseudocode as Algorithm 1.

Algorithm 1 Town-level route planning

1: Preprocess RNDF to build graph G
2: **while** running **do**
3: Localize robot in G
4: Get obstacle (blockage) data from local perception level
5: Add blockages to G
6: Search G for best path to next checkpoint
7: Send initial path segment to street-level planner
8: **end while**

Once the graph is constructed, we find paths using the standard A* search algorithm.[6] In this search, links have a cost proportional to their length, lane change links have a linear penalty, and exits also have a small penalty. The search starts at the closest location in the RNDF to our current global position, as supplied by our GPS/INS unit, and runs to the next checkpoint the vehicle is required to pass through. We then traverse the resulting path to find the first stop sign or exit.

During a run, it is possible to modify the graph in two ways. New waypoints can be added, increasing the accuracy of the RNDF. Also, lanes can be marked as blocked (using information from the local perception level), which would cause a different plan to be found during subsequent iterations.

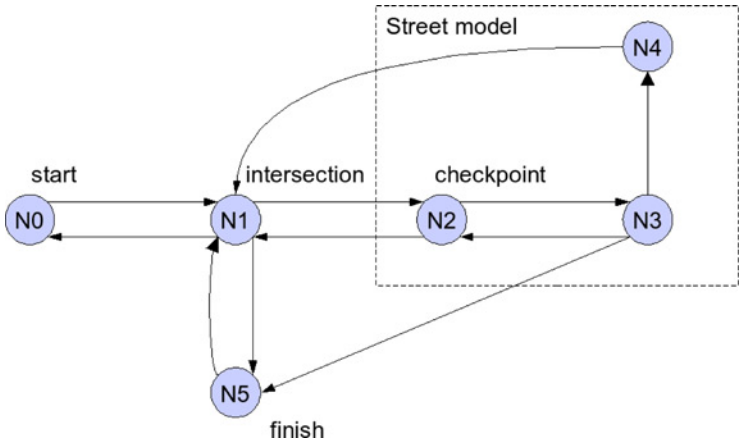

Fig. 9 Town model – a connected graph constructed from the RNDF.

D. Street Level

The street-level planner is the component in our system that plans for lane changes and intersections. The role of this layer is to plan ahead along the current street, avoiding large obstacles such as cars, until the next stop sign or exit (which was passed down by the route planner). Plans are constructed in a 1.5-dimensional, discrete space. That space has distance down the current segment in 3-meter increments as one dimension, and lane number as a second dimension. Each discrete cell is either free or contains an obstacle passed up from the local level. Every obstacle is assumed to have a discrete velocity along the road, allowing our street-level plan to anticipate the actions of other vehicles. The town map passes in the rough shape of the road ahead in terms of the curvature of each segment. This can be used to reduce speed ahead of sharp corners. Fig. 10 illustrates the representation. Planning proceeds with a simple model of other vehicles using Real-Time Dynamic Programming.[7]

The local motion planner in our system takes commands in the form of a series of waypoints specifying a trajectory. The street-level planner gives commands in terms of where to change lanes and a maximum speed. We have a simple system that maps between these two types of commands.

The mapping takes the lane center-line generated from the town-level RNDF map, and the detected lane information. The map center-line is shifted laterally until it is aligned with the lanes detected by the car's vision system. We then loop along this corrected center-line until the distance for the next street-level command is reached, passing the center-line down as the trajectory, and marking the maximum speeds as appropriate. If a lane change is detected then the trajectory is offset as appropriate to place it in the correct lane. Fig. 11 illustrates the waypoints generated by the street-level planner during a lane change.

At a corner, there is no trajectory information passed down. Rather, the street-level planner provides the low-level planner with the next waypoint in the town-level map, and the low-level planner calculates a smooth trajectory for the turn.

E. Local Level

1. *Static Obstacle Detection and Representation*

For reliability and robustness, we have chosen to use a standard grid map data structure often used in robotics to represent obstacles in the environment.[8,9] We closely follow the obstacle detection technique used by Thrun *et al.*[9] in the last challenge, in which the scans from downwards looking lasers are projected into the vehicle's local coordinate frame, resulting in a 3D point cloud for each laser. Laser points are grouped into 2D cells which combine to make a surface grid. The traversability of a cell is classified by vertical height differences between laser points within a cell. A cell with a height difference greater than some threshold is deemed not traversable. Unlike Thrun *et al.*, we did not need the additional machine learning layer required to compensate for small inaccuracies in pose.

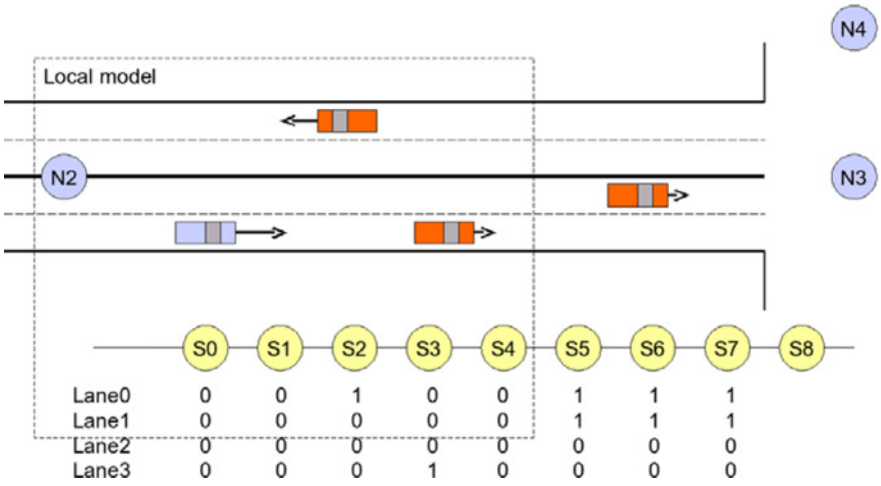

Fig. 10 Street model – a 1.5 dimensional, discrete space with distance down the current segment in three meter increments as one dimension, and lane number as the extra half dimension.

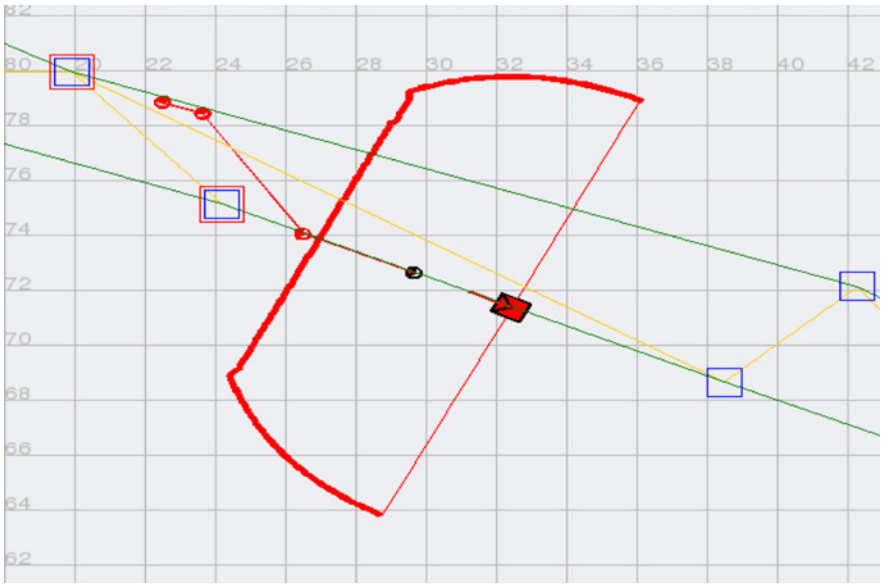

Fig. 11 Screen capture of our visualization tool as the car approaches a lane change. The blue boxes are RNDF waypoints. The red boxes are RNDF waypoints that the car plans to travel through. The small red circles are low-level waypoints for the local navigation component to follow. The large red D shape is the laser return. Note that once the local waypoints are past the last RNDF waypoint in the current lane, they are offset into the neighboring lane to cause the car to change lanes.

We found that accurate timestamps recorded using the hard real-time QNX operating system for both the navigation solution and laser scans resulted in no such error.

An example of a traversability map is shown in Fig. 12. This map is a display version of the grid's data structure which is provided to the navigation and path-planning components.

2. Lane Detection

Unlike the first two grand challenges, abstract objects such as lane markings must be detected and stored for use in high-level decision making. Although reflective paint can assist lasers in detecting lane markings, we chose to use visual sensors in case poor or non-reflective lane markings were present. However, detection of lane markings in image data is not a trivial task. The difficulty is mainly due to the lack of unique models, poor quality of lane markings due to wear, occlusions due to the presence of traffic, and complex road geometry.

The potential pitfalls when using image data led us to the conclusion that any method using a strong model of the road (lane) will eventually fail. Thus, weak models must be applied. Lane detection is generally based on the localization of a specific pattern (the lane markings) in the acquired image and can be performed with the analysis of a single still image.

The method for lane marking detection employed in our paper is based on the Generic Obstacle and Lane Detection (GOLD) implementation.[10] In order to fit the lane model to the acquired road images, geometrical image warping is performed with Inverse Perspective Mapping (IPM). The IPM technique projects each pixel of a 2D perspective view of a 3D object and remaps it to a new position, constructing a new image on an inverse 2D plane. Conversely, this will result in a bird's-eye view of the road, removing the perspective effect. Each pixel represents the same portion of the road, allowing a homogeneous distribution of the information among all image pixels. To remove the perspective effect, it is necessary to know *a priori* the specific acquisition conditions (camera position, orientation, optics) and the scene represented in the image (the road).

The resolution of the remapped image has been chosen as a trade-off between information loss and processing time. At the moment we are using the red channel of an RGB image as it provides the highest contrast between the lane and pavement color. The approach was further chosen, as the camera selected on our vehicle is a 3CCD camera,

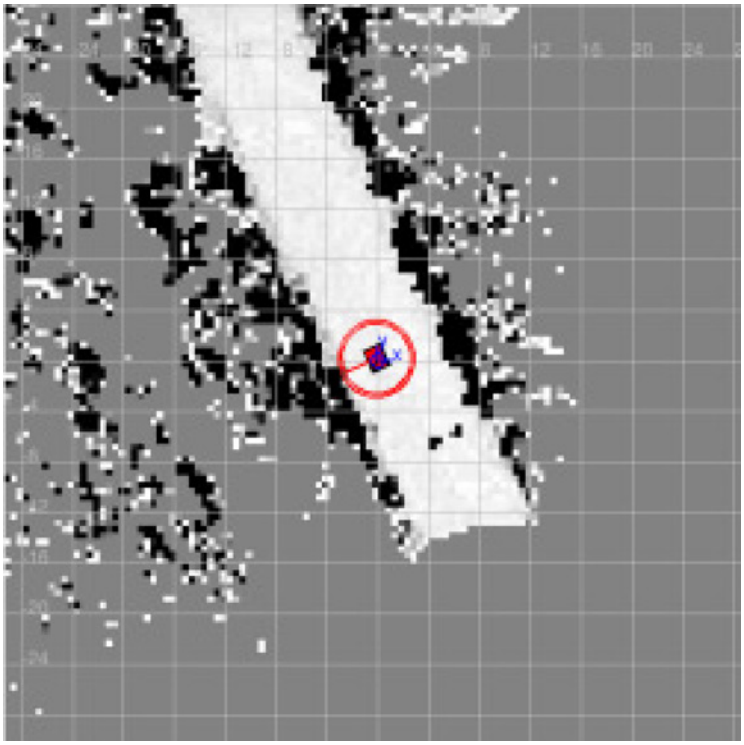

Fig. 12 Snapshot from the GUI software component illustrating the traversability map. White indicates traversable, black indicates an obstacle larger than 30 cm, and shades of gray indicate objects 0–30 cm. Cell sizes are 40 cm. The red rectangle represents the vehicle.

enabling retrieval of each image channel without any compression losses from a Bayer pattern. Another potential space to be used is the C1C2C3 space, which gives marginally better results but at the cost of computational effort.

As mentioned above, our mapping assumes that the road is flat. This assumption does not hold in most cases, so we use a vertically mounted laser to detect the relative pitch of the camera and the road.

Lane marking detection is performed on the remapped image by applying a lane model which stipulates that a road marking is represented by a predominantly vertical bright line (lane marking) of constant width surrounded by a darker region (the road). Thus, the pixels belonging to a road marking have a brightness value higher than their left and right neighbors at a given horizontal distance. A vertical edge in an image conforms similarly to the same principle; the intensity difference between neighboring pixels must be over a threshold to be validated. This is detected by performing a Sobel Edge Operation[11] on the red channel of the image. An exhaustive search across each row of the image will produce potential lane marking candidates where the match probability can be measured with the edge quality, i.e. the difference in intensity.

It is assumed the center lane marking consists of a double yellow line and the other lane markings consist of a single white line. Rows in an image are sequentially analyzed. Pixels within a row that are classified as part of a lane marking form an "observation" for a spatial, recursive filter which propagates vertically through the image. A pixel is deemed an observation if there are similar adjacent pixels that conform to a fixed lane marking width when grouped together. Propagation of the filter occurs using a transition model which assumes that lane markings are more likely to have a low degree of curvature and that there is a maximum degree of curvature which cannot be exceeded. An example of detected lanes is shown in Figs. 13(c) and 14.

Finally, the virtual lane trajectory is calculated from the detected center lane marking additionally verified against the other lane markings (width of lane defined approximately by the RNDF) and represented as a polyline. It is then communicated to the path-planner and incorporated as a constraint. The detected center lane marking is propagated between the image frames using the integrated solution of the onboard inertial measurement unit. The position of

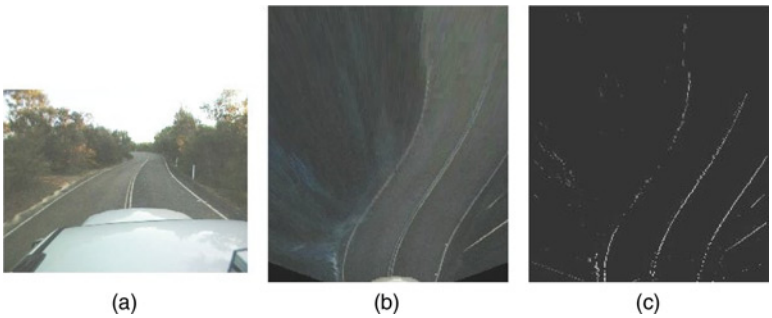

Fig. 13 a) Original image, b) IPM image, c) Detected lane marking segments.

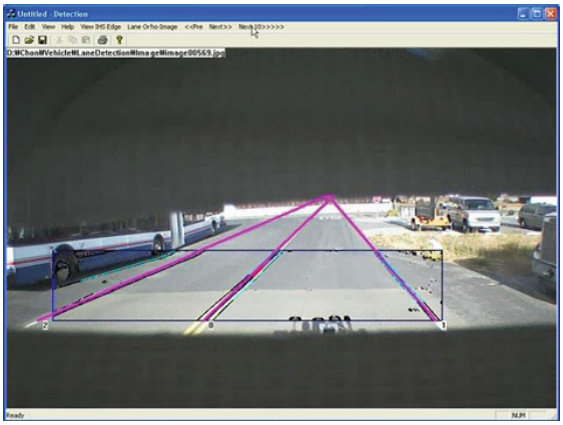

Fig. 14 Detected lanes: pink lines represent the observed lanes projected onto the image in perspective. The box represents the area of the image in which we search for lanes.

the propagated center lane marking is then used as a seeding point for the next iteration of the filter and contributes to the robustness and improved efficiency of the algorithm. Mean processing time for detecting the initial lane trajectory is 70 milliseconds, but once the lane is detected and propagated between frames the processing time drops to 5 milliseconds.

3. Local Motion Planning

The trajectory and speed markers generated by the street-planning level provide goal information to the final component of our planning system, the low-level motion planner. The speed markers are treated as point goals by the system. The responsibility of this component is to control the car to visit the goal waypoint sequence while avoiding obstacles and staying close to the provided trajectory (which represents the lane information). This task is a standard type of problem in robot motion planning, and this useful decomposition is a strength of our layered planning system design. However, in our case we are planning for a car-like robot, and this involves differential constraints. Motion planning under differential constraints remains an extremely challenging problem in the literature.

Two main categories of algorithms for motion planning are the well- established combinatorial methods, and the more recent sampling-based methods.[12,13] Because combinatorial methods in general do not consider nonholonomic constraints or any type of dynamics, the current best options are sampling-based methods. We chose to use a variant of the popular Rapidly-Exploring Random Trees (RRT) algorithm.[13,14] RRTs have been used successfully with real robots and demonstrate acceptable real-time performance.[15,16] We modified the basic algorithm to incorporate lane following and to respect velocity and acceleration bounds. We list pseudocode as Algorithm 2 and discuss the algorithm in detail in this section.

Algorithm 2 RRT-based motion planning

```
1: while running do
2:    Get new waypoints
3:    if no waypoints then
4:       Send stop action to actuator controllers
5:    end if
6:    Get obstacle data
7:    Get trajectory data
8:    Initialize tree to previous best path
9:    for i = 1 to desired number of expansions do
10:      With probability p:
11:         Randomly choose node n from tree
12:         Randomly choose safe control values
13:         Predict new state s
14:      With probability 1 − p:
15:         Randomly choose node n from tree
16:         Use heuristics to choose safe control values that guide robot toward goal
17:         Predict new state s
18:      Compute distance to nearest obstacle from s
19:      Compute weighted cost of leaf-to-root path from s
20:      Compute both optimistic and pessimistic cost estimates to the next goal
21:      if the total optimistic cost for this node is greater than the least pessimistic cost for any node then
22:         Prune path
23:      else
24:         Add s to tree
25:      end if
26:   end for
27:   Store best-cost path
28:   Send initial control action in best path to actuator controllers
29: end while
```

The standard RRT algorithm searches for a collision-free path between start and goal states. It searches by iteratively expanding a search tree until it finds the goal. Here the start state is current robot position, and goal states are waypoints provided by the street-level planner. We expand the tree by forward integration of control values from a randomly-chosen node in the tree. The control values respect velocity and acceleration bounds and are either determined by heuristics that lead towards the goal or are chosen randomly. We prune unfavorable leaves in the tree using a weighted cost function that takes into account Euclidean distance to goal, distance to obstacles, proximity to the supplied trajectory, and path length. Because of this pruning, our sampling is biased towards favorable regions of configuration space. Obstacles are sensed using the laser scanner and provided to the planner in a grid representation. The algorithm performs a fixed number of expansions and then returns the best-valued control action (desired speed, desired steering angle), which is then executed by the low-level hardware controllers. We interleave this planning with a check for new waypoints and current sensor data. Since waypoints do not generally change radically between calls to the planner, we store the last best path for use in initializing the tree in the next iteration. In fact, the vehicle rarely reaches any given waypoint before the street-level planner provides a new set (receding-horizon control).

We performed preliminary evaluation in simulation using the Player/Gazebo environment.[c] The CPU time spent choosing a control action ranged from 20 to 100 milliseconds on a standard workstation running both the simulator and the planner. Similar performance was observed during online hardware testing with the vehicle. A snapshot of the motion planner execution in our GUI is shown in Fig. 15.

[c] http://playerstage.sourceforge.net/

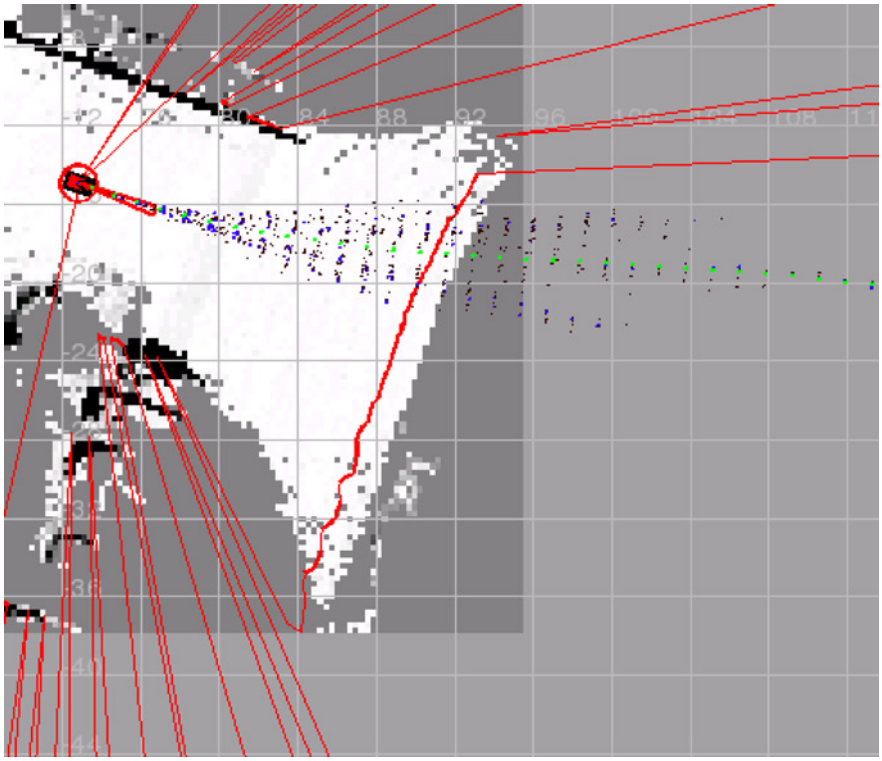

Fig. 15 The motion planner execution in our GUI. The vehicle is drawn as a rectangle, search tree nodes are drawn as pixels emanating from the vehicle. The red lines represents returns from one of the lasers projected in 2D.

Because we expand the RRT tree using safe control values, the resulting path should not violate the vehicle's dynamic constraints. An obvious downside to this biased sampling is that we explore a restricted region of configuration space. We have not observed problems in finding good paths in our simulation experiments, however, most likely due to useful heuristics and relatively simple obstacles.

IV. Results and Analysis

To evaluate our system, we performed experiments in a real urban environment. Our objective was to assess the robustness of our system in unfavorable GPS conditions. To do this, we performed navigation runs around a 560-meter loop flanked by buildings, large trees, ditches, and other obstacles. We also demonstrated high-level behaviors such as overtaking parked vehicles and proper queuing behavior approaching an intersection. An aerial view of this test track is shown in Fig. 16.

We report results from two navigation runs, alternating travel direction (clockwise/counter-clockwise) around the loop. The runs consisted of ten laps each and evaluate the complete system including lane detection. Each run used the following protocol. We began with the system in *pause* mode at a common starting point (see Fig. 16g) and loaded a mission definition file (MDF) that encoded the desired travel direction and number of laps. We manually switched the system into *run* mode and continued until a failure requiring manual intervention. After a failure, we manually switched the system to *pause* mode, corrected the failure, and switched back to *run* mode. We continued in this way until we completed the desired number of laps. Raw data from all sensors (including camera images) was stored in a log file for analysis. Failures consisted of 1) a hardware failure in the power steering system requiring turning the car's engine off, then back on 2) a false obstacle requiring us to manually drive the car past 3) a bad turn requiring us to manually drive the car back onto the road, or 4) a planning failure where the car failed to make a turn and was unable to correct itself via a u-turn.

Fig. 16 Aerial view of test track. The transparent yellow line represents the 560-meter square loop in which we tested our vehicle. The letters indicate correspondence with the lower figures as seen from the vehicle. Note that GPS coverage is minimal along the east and north segments of the loop with areas where no satellites are observed.

Table 1 Results summary.

	Clockwise	Counter-clockwise	Total
Laps	10	10	20
Elapsed time (minutes)	51	53	104
Distance traveled (km)	5.6	5.6	11.2
Most consecutive laps without failure	7	4	7
Max speed (kph)	22	22	22
Failures			
Power steering	1	2	3
Bad turn	0	1	1
False obstacle	2	6	8
Planning	1	0	1
Total	4	9	13

We provide a summary of our results in Table 1. The majority of errors were due to hardware problems (power steering failures) or false positives in our obstacle detection subsystem. We had only one unrecoverable failure due to a bad turn, and one due to a planning error in 80 turns. Our longest stretch without a software failure was seven laps.

We present a detailed analysis of these data in the following subsections. First we investigate lane detection accuracy and then combine this with navigation information to evaluate the system as a whole.

A. Perception

The task of our lane detection module is to determine the position of the center of the road, marked by a double yellow line, relative to the robot. This localization compensates for errors in global position estimation. Fig. 17 shows output from a typical run. To establish ground-truth position, we instrumented the test track with special features observable in the logged laser data. These features are placed at known locations around roughly half of our test track between the labels (b),(d),(a), and (c) in Fig. 16.

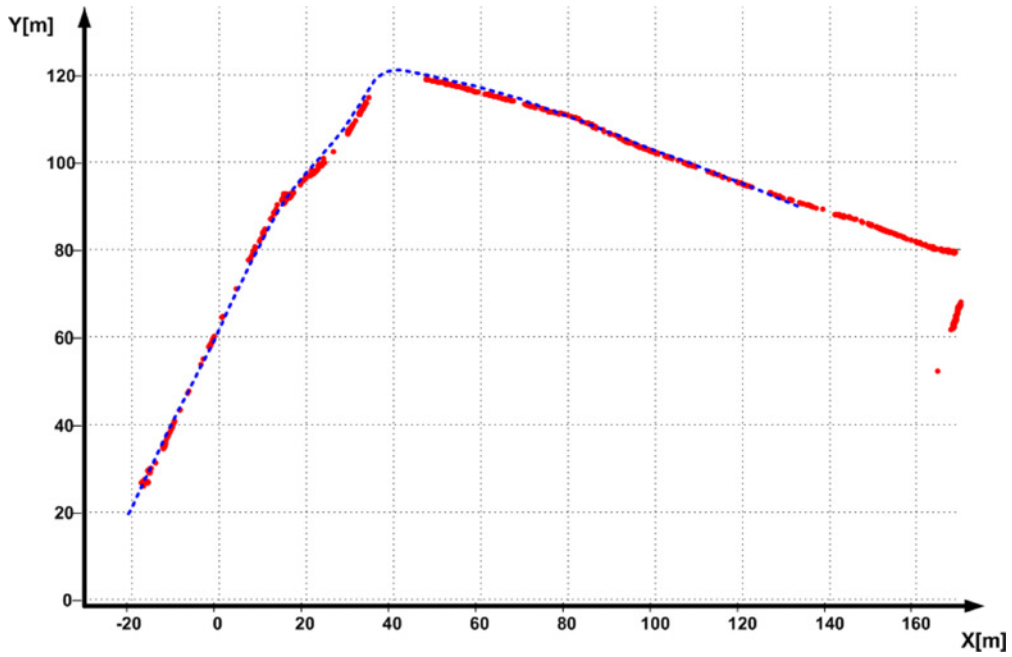

Fig. 17 Comparison between the actual center line (blue dashed line) and the lane extracted from vision (red lines).

By manually checking the chosen center marking against stored camera images, we found only 0.5% false positives. Mean frames between false negatives was approximately 10. Our system is tolerant to a reasonable number of false negatives since we simply reuse the most recent measurement. As long as we sense a lane relatively soon after leaving an intersection, false negatives are not a problem. Since the number of false positives was low, we take this as evidence that our lane detection module works exceptionally well.

B. Motion Planning with Lane Detection

The first challenge in analyzing full system performance is to quantify the magnitude of navigation error. From ground truth data, we can compare the actual position of the car to the estimated global position received from GPS. Fig. 18 shows that our estimated position changes rapidly and can be up to 2.1 m away from the true position. This error is severe. Relying on GPS alone results in frequently driving completely off the road. Without vision-based lane detection for correction of the navigation solution, we rarely complete one full loop without an unrecoverable failure.

This large absolute error in position estimate can lead to two types of problems: 1) lateral error that leads us to drive out of our lane, and 2) longitudinal error that leads to difficulty in making turns at intersections. Type 2 errors are uncorrected in our current system and we must rely on obstacle avoidance to make turns safely. Type 1 errors are addressed by our lane detection system.

Having established the accuracy of our lane detection system, a final measure of system performance is possible simply by analyzing how well the car tracks the adjusted lane line provided to the path planner. Results for five loops are shown in Fig. 19.

Here we see that the car stayed on its target line with a mean error of 3 cm and a standard deviation of 66 cm. This result is striking when compared to the GPS/IMU trajectories without visual lane correction shown in Fig. 20 with errors of up to 2 m in the actual navigation solution. The larger lane tracking errors can arise from three possible sources: 1) obstacle avoidance, 2) hardware failure, and 3) entering/leaving intersections.

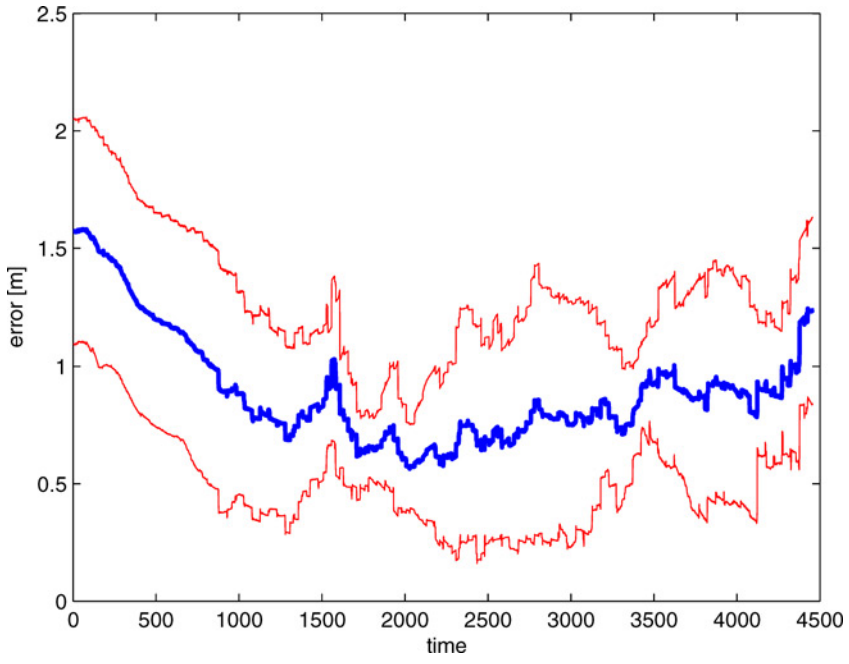

Fig. 18 The absolute error in the global navigation solution (blue line) and 1 standard deviation (red lines) without visual lane correction.

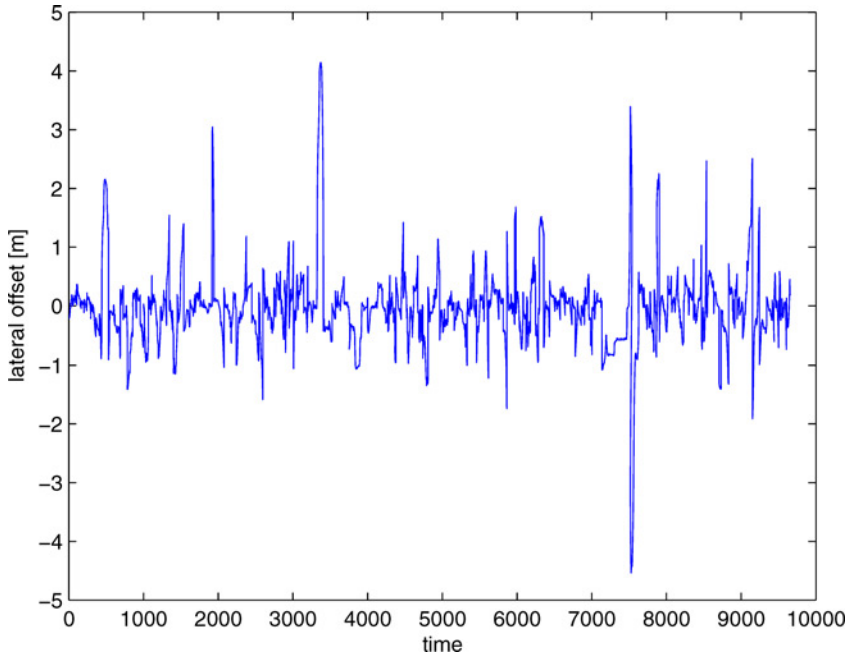

Fig. 19 Error between desired and actual position over five loops.

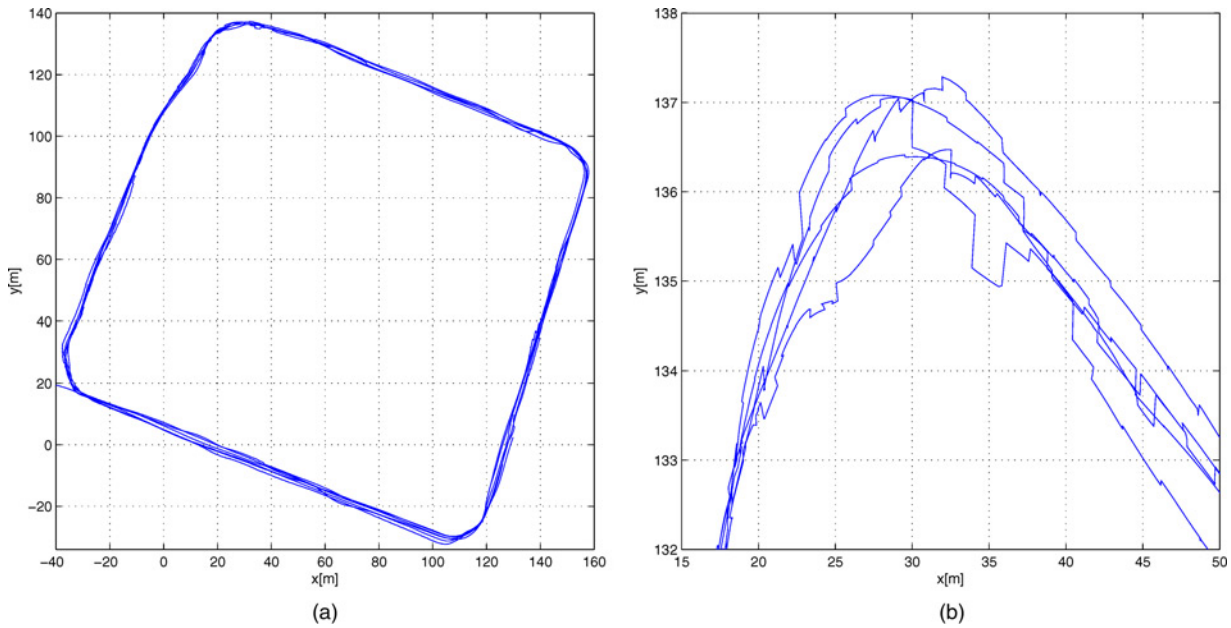

Fig. 20 (a) Overlay of GPS/IMU trajectories for five loops. (b) Magnified view of the top corner in (a).

V. Discussion

Of the failures reported in our results, the majority were due to false obstacles and hardware failures in the power steering system. These are serious problems that require manual intervention for recovery. However, we do not

feel these are fundamental flaws and we believe both will be positively resolved in the near future. We expect that our vehicle will then exhibit reliably good performance on novel stretches of road using lane detection for lateral correction.

This lateral correction does not address problems at intersections, however, since it does not compensate for longitudinal error in position estimates. It appears necessary to integrate information from lasers to give a better local estimate of the geometry of an intersection. This information would help us to make smoother turns.

Finally, although we have shown robust navigation runs, we have yet to fully validate the intelligent high-level planning capabilities of our system in handling four-way intersections and operating among traffic in general. We did show initial positive results with queuing behavior and overtaking parked cars, and hope to focus on empirical evaluation in this area in future research.

Acknowledgments

Thanks to everyone from UC Berkeley and the Sydney team who contributed to the project. Thank you to Alex Brooks and the Orca developers. Also, a special thanks to Jonathan Sprinkle for the long hours.

This work was supported in part by Rio Tinto, Komatsu, Toyota, Chess at Berkeley, ZeroC, and Advantech. Additional inkind support was provided in the form of discounts on equipment from the following manufacturers: SICK, NovAtel, and Honeywell.

The University of Sydney and the University of Technology Sydney are supported by the ARC Centre of Excellence program, funded by the Australian Research Council (ARC) and the New South Wales State Government.

National ICT Australia is funded by the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

References

[1]"Special Issue on the DARPA Grand Challenge, Part 1," *Journal of Field Robotics*, Vol. 23, 2006, pp. 461–652. doi: 10.1002/rob.20142

[2]"Special Issue on the DARPA Grand Challenge, Part 2," *Journal of Field Robotics*, Vol. 23, 2006, pp. 655–835. doi: 10.1002/rob.20154

[3]Brogdon, J., Dunlap, R., Dyson, P., Martin-de-Nicolas, A., Martin-de-Nicolas, J., Martin-de-Nicolas, J., McCauley, D., Miner, D., O'Quin, J., Polkowski, S., and Roever, J., "Austin Robot Technology, Inc. Technical Paper," Tech. rep., http://www.darpa.mil/grandchallenge05/TechPapers/Austin_Robot_Technology.pdf, DARPA Grand Challenge, 2005.

[4]DARPA, "E-Stop Guidelines," Tech. rep., http://www.darpa.mil/grandchallenge/docs/E-Stop_Guidelines_042307.pdf, DARPA Grand Challenge, 2007.

[5]Albus, J., "4D/RCS: A Reference Model Architecture for Intelligent Unmanned Ground Vehicles," *SPIE 16th Annual International Symposium on Aerospace/Defense Sensing, Simulation and Controls*, 2002.

[6]Hart, P., Nilsson, N., and Raphael, B., "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics SSC4*, Vol. 2, 1968, pp. 100–107.

[7]Barto, A. G., Bradtke, S. J., and Singh, S. P., "Learning to Act using Real-time Dynamic Programming," *Artif. Intell.*, Vol. 72, No. 1-2, 1995, pp. 81–138.

[8]Elfes, A., "Occupancy Grids: A Stochastic Spatial Representation for Active Robot Perception," *6th Conference on Uncertainty in AI*, 1989.

[9]Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.-E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., and Mahoney P., "Stanley: The Robot that Won the DARPA Grand Challenge," *Journal of Field Robotics*, Vol. 23, pp. 661–692. doi: 10.1002/rob.20147

[10]Bertozzi, M. and Broggi, A., "GOLD: A Parallel Real-time Stereo Vision System for Generic Obstacle and Lane Detection," *IEEE Trans. on Image Processing*, Vol. 7, pp. 62–81. doi: 10.1109/83.650851

[11]Duda, R. and Hart, P., *Pattern Classification and Scene Analysis*, John Wiley and Sons, 1973.

[12]Latombe, J., *Robot Motion Planning*, Kluwer Academic Publishers, 1991.

[13]LaValle, S., *Planning Algorithms*, Cambridge University Press, 2006.

[14]Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L. E., and Thrun, S., *Principles of Robot Motion: Theory, Algorithms, and Implementations*, MIT Press, 2005.

[15]Bruce, J. and Veloso, M., "Real-Time Randomized Path Planning for Robot Navigation," *IROS*, 2002.
[16]Ladd, A. and Kavraki, L., "Fast Tree-based Exploration of State Space for Robots with Dynamics," *Algorithmic Foundations of Robotics VI, Springer STAR*, Vol. 17, 2005, pp. 297.
doi: 10.1007/10991541_21

Christopher Rouff
Associate Editior