2008

A Practical Approach to Robotic Design for the DARPA Urban Challenge

Benjamin J. Patz

Yiannis Papelis
Old Dominion University, ypapelis@odu.edu

Remo Pillat

Gary Stein

Don Harper

Original Publication Citation

Patz, B. J., Papelis, Y., Pillat, R., Stein, G., & Harper, D. (2008). A practical approach to robotic design for the darpa urban challenge. *Journal of Field Robotics, 25*(8), 528-566. doi:10.1002/rob.20251

A Practical Approach to Robotic Design for the DARPA Urban Challenge

Benjamin J. Patz
Coleman Technologies, Inc.
Orlando, Florida 32801
e-mail: bpatz@ctiusa.com

Yiannis Papelis
Virginia Modeling Analysis and Simulation Center
Old Dominion University
Suffolk, Virginia 23432
e-mail: ypapelis@odu.edu

Remo Pillat, Gary Stein, and Don Harper
College of Engineering and Computer Science
University of Central Florida
Orlando, Florida 32816
e-mail: rpillat@cs.ucf.edu, gstein@mail.ucf.edu, harper@cs.ucf.edu

Received 29 January 2008; accepted 18 June 2008

This article presents a practical approach to engineering a robot to effectively navigate in an urban environment. Inherent in this approach is the use of relatively simple sensors, actuators, and processors to generate robot vision, intelligence, and planning. Sensor data are fused from multiple low-cost, two-dimensional laser scanners with an innovative rotational mount to provide three-dimensional coverage with image processing using both range and intensity data. Information is combined with Doppler radar returns to yield a world view processed by a context-based reasoning control system to yield tactical mission commands forwarded to traditional proportional-integral-derivative (PID) control loops. As an example of simplicity and robustness, steering control successfully utilized a relatively simple follow-the-carrot guidance approach that has been successfully demonstrated at speeds of 60 mph (97 km/h). The approach yielded a robot that reached the finals of the Urban Challenge and completed approximately 2 h of the event before being forced to withdraw as a result of a global positioning system data failure.

1. INTRODUCTION

The Urban Challenge is the third in a series of competitions launched by the Defense Advanced Research Projects Agency (DARPA) with the goal of developing technology to keep warfighters off the battlefield and out of harm's way. The specific objective of the Urban Challenge was to develop a robot capable of autonomously navigating a typical urban environment at speeds up to 30 mph (48 km/h). Urban scenarios involved other manned vehicles as well as robots traversing the same course at the same time and resulted in robot-on-robot autonomous decision-making challenges. The final event of the competition took place in Victorville, California, on November 3, 2007, but this event was a culmination of 18 months of work and numerous other formal qualification procedures. TeamUCF and the Knight Rider robot successfully passed these qualification procedures to make it to the finals of the Urban Challenge.

1.1. Urban Challenge Overview

The Urban Challenge program was announced in May 2006, and proposals from interested teams were solicited shortly thereafter. Successful proposals were divided into two categories: 11 Track A teams received $1 million in supporting funding from DARPA, and 78 Track B teams were on their own. TeamUCF was a Track B team. Track A teams had to meet several programmatic milestones, but any team advancing in the competition had to submit a comprehensive technical report and pass an on-site visit by DARPA. Site visits were conducted in June and July 2007, and 35 semifinalists were selected in August. Semifinalists were eligible to participate in the National Qualifying Event (NQE) in Victorville, California (Figure 1), during the last 2 weeks of October 2007. The NQE consisted of a series of rigorous vehicle tests from which 11 finalist teams were selected. These finalists participated in the final event on November 3, 2007, with the winner of the competition announced the following day.

The overall urban driving objective as defined by DARPA was to demonstrate an autonomous robot's ability to complete a series of driving missions in traffic, over the course of 6 h, while obeying California driving rules, utilizing a moderate level of a priori information associated with the road network, but being expected to deduce any missing information. Some key observations can be made with respect to this definition, some of which were clearly specified in DARPA-provided rules, whereas others simply became apparent over the stages leading up to the final event:

- Robots were limited to street-legal motor vehicles, modified for autonomous operation.
- Autonomous operation meant no real-time interaction with the robot except for a remote control safety (E-stop) system, which could pause or completely disable the robot.
- Urban environment consisted of typical U.S. streets with one-way and two-way single-lane roads, multilane roads, traffic circles, intersections with zero or more stop signs, and parking lots (zones). No stop lights were encountered. Surprisingly, a modest amount of off-road driving was required in the final event.
- Traffic vehicles meant that following, passing, avoidance, and stop sign precedence behavior was required.
- A priori information consisted of a route network definition file (RNDF) that defined road segments/lanes, provided by a sparse, but accurate, collection of latitude/longitude "way points" and a way point-to-way point connectivity graph (although connectivity was not guaranteed because roads could be blocked by design or by accident). The RNDF also provided stop sign locations. Nominally, the RNDF was provided a day or more before any test.
- The driving mission was to traverse a certain set of way points (i.e., checkpoints) in a given order as defined in a mission definition file (MDF). Nominally, the MDF was provided minutes before a test.

1.2. TeamUCF

TeamUCF's Knight Rider robot was initially conceived more than 3 years ago at the beginning of the 2005 DARPA Grand Challenge event. With the inception of the DARPA Urban Challenge, TeamUCF built on the existing capabilities of the Knight Rider robot (Harper et al., 2005), augmented as necessary to meet the specific mission objectives of DARPA. The three-member team that participated in the 2005 event was expanded only slightly for the 2007 Urban Challenge (five core team members) with the inclusion of an

Figure 1. NQE site in Victorville consisting of several test areas.

industry partner, Coleman Technologies, Inc. (CTI). CTI is a system engineering firm specializing in real-time guidance, navigation, and control as well as products associated with global positioning system (GPS) measurements in urban environments. CTI provided funding, technical support, and overall team leadership for TeamUCF. TeamUCF was kept small, partly by design but mainly as a natural result of the team's goal to reuse much of the 2005 robot hardware and a clearly stated objective to implement only those systems necessary to meet stated DARPA Urban Challenge objectives.

1.3. Overall Project Approach

TeamUCF's overall approach to this challenge was to maximize its limited resources. The basic robot control hardware from the 2005 robot was reused. The sensor suite was an enhanced version of the 2005 robot's sensor suite, which had proved to be very robust and which also turned out to be a sensor suite used by many of the participating teams. Three major weaknesses with the 2005 robot were specifically addressed:

- A relatively poor GPS system was replaced with a highly capable RT3000 GPS/inertial navigation system (INS) from Oxford Technical Solutions. This decision was a key to TeamUCF's initial success, but confidence in the GPS/INS ultimately led to an unrecoverable failure in the final event.
- A relatively poor simulation model was replaced with a real-time simulator running actual robot software and could operate with full hardware-in-the-loop capability to allow any system element to be real or simulated.
- A more focused preevent testing strategy was employed that allowed the team to have a clear understanding of the capabilities and

limitations of the robot prior to participation in the NQE. This level of knowledge allowed the team to make software changes between tests at the NQE and was perhaps the single most important factor in the team's success.

Resource constraints limited the team's ability to invest in sophisticated three-dimensional (3-D) laser scanners, so the team opted for an investment in an innovative rotating laser scanner system designed by one of the authors (Pillat). The limited reliance on sophisticated third-party systems, with capable but inherently unmodifiable software, proved to be a fundamental advantage for TeamUCF.

Simulation modeling allowed all major software elements to be tested prior to robot integration, but there was no substitute for actual robot testing, and the team spent as many hours as possible testing the robot hardware. Unfortunately, testing a full autonomous automobile-sized robot at speed poses numerous safety issues, and TeamUCF was forced to settle on relatively small test areas whose access could be controlled.

The system design approach could be considered a cross between "requirements based" and "capabilities based" in that overall Urban Challenge objectives flowed down to scenarios (Figure 2) and subsequently to overall system-level capabilities, but detailed subsystem performance requirements were not derived from these. Rather, because subsystems were effectively selected at the start of the project, the challenge for TeamUCF became one of determining "how" to meet a specific objective with a given system, rather than what system would be best for meeting a particular objective.

In approaching the software functional design, the team followed these basic principles:

- Safety was of primary importance. The goal was to provide a system that would protect the Knight Rider from collisions and kinematic limits, protect other robots from collision or perceived collision, and protect obstacles.
- Mission completion was of secondary importance. The goal was to complete as much of the provided mission as possible, potentially skipping checkpoints if the robot determined that they were not achievable.
- Legality was of tertiary importance, meaning that the system's software was allowed to violate rules if there was no other way to meet an objective.
- Speed was of least importance. Despite its relative lack of importance, it turned out that the Knight Rider was one of the quickest robots at the NQE.

2. ROBOT VEHICLE

The Knight Rider robot is a 1996 Subaru Outback Legacy (Figure 3) with minimal modifications. Key performance parameters for this robot are provided here:

- 4.8-m overall length with mounting brackets
- 2.0-m overall width with mounting brackets
- 2.6-m wheelbase
- 5.5-m turning radius
- Speed: −2.2–13.5 m/s (−5–30 mph, DARPA restricted)
- Axial acceleration: ∼3.5 m/s^2 (practical limits for comfortable driving)
- Lateral acceleration: ∼2 m/s^2 (practical limits for comfortable driving)

Adopting this vintage vehicle prevented the use of drive-by-wire or other sophisticated integration into an automobile control system. This limitation turned out to be of no impact on robot performance. Actuators were designed to control existing robot hardware in a manner analogous to that of a human operator. For example, the steering servo, mounted along the robot centerline, controlled the steering wheel with a belt system similar to the way a driver would control that system. This system easily allowed both robotic operation as well as driver operation. Because of safety concerns, most testing prior to the NQE was conducted with a driver in the vehicle. Servo torques and belt slippage were adjusted to allow driver override even in the event of full system failure. Throttle and brake actuators were mounted under the passenger seat and similarly provide fail-safe operation. In particular, the brake actuator causes the vehicle brake to be depressed and was in the "always on" position via a spring mechanism. The brake was "released" via a pneumatic actuator that, should it fail, would cause the brake to return to the depressed position. Turn signal integration was accomplished via the vehicle's existing wiring infrastructure.

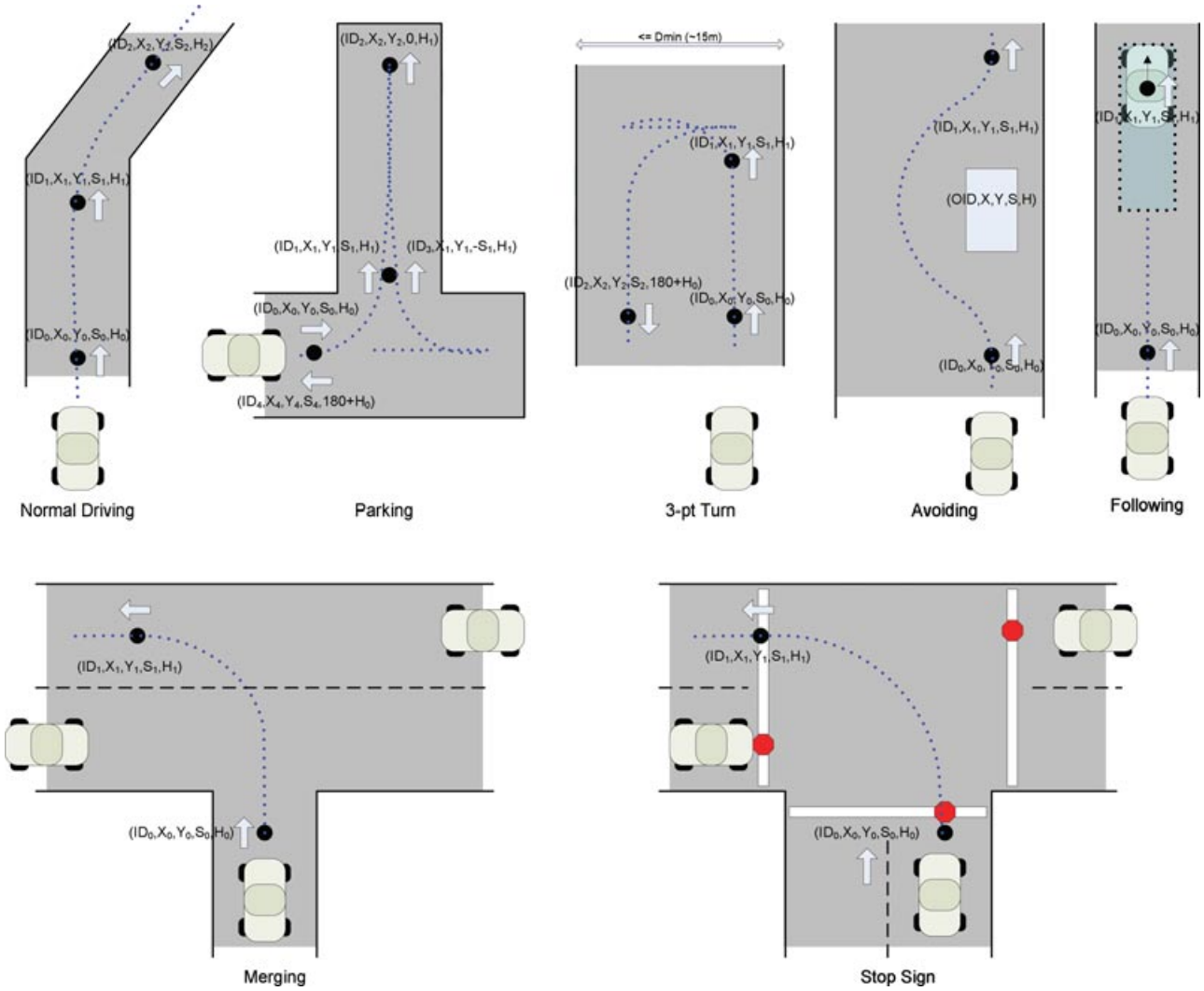

Figure 2. Basic driving scenarios (not all cases shown).

Own-state estimation (position, speed, and heading as well as full robot attitude) was provided by a differentially corrected RT3000 GPS/INS from Oxford Technical Solutions. Previous experience indicated that this was an area where commercial systems outperformed developed software. The integrated INS provides high-quality measurements, including obscured-sky speed measurement, in environments where the GPS alone struggles; lateral acceleration in a horizontal direction without the need to zero the accelerometer; and roll/pitch/yaw measurements that are accurate during continuous turns. GPS/INS data were available to other systems at a 100-Hz data rate, although low-level control systems operated only at 20 Hz and the highest sensor data rate was 35 Hz.

SICK LMS291 laser scanners mounted on a forward and rear mounting bracket and rotating laser scanners mounted to the top rack provide range, angle, and intensity information on obstacles as small as traffic cones. The sensors provide information only for the leading edge of obstacles, but after multiple looks from varying angles, obstacle geometry is refined. Scanner pointing direction and type were selected to optimize forward sector coverage. This approach also provided 100% overlap in coverage directly in front of the robot, which proved to be

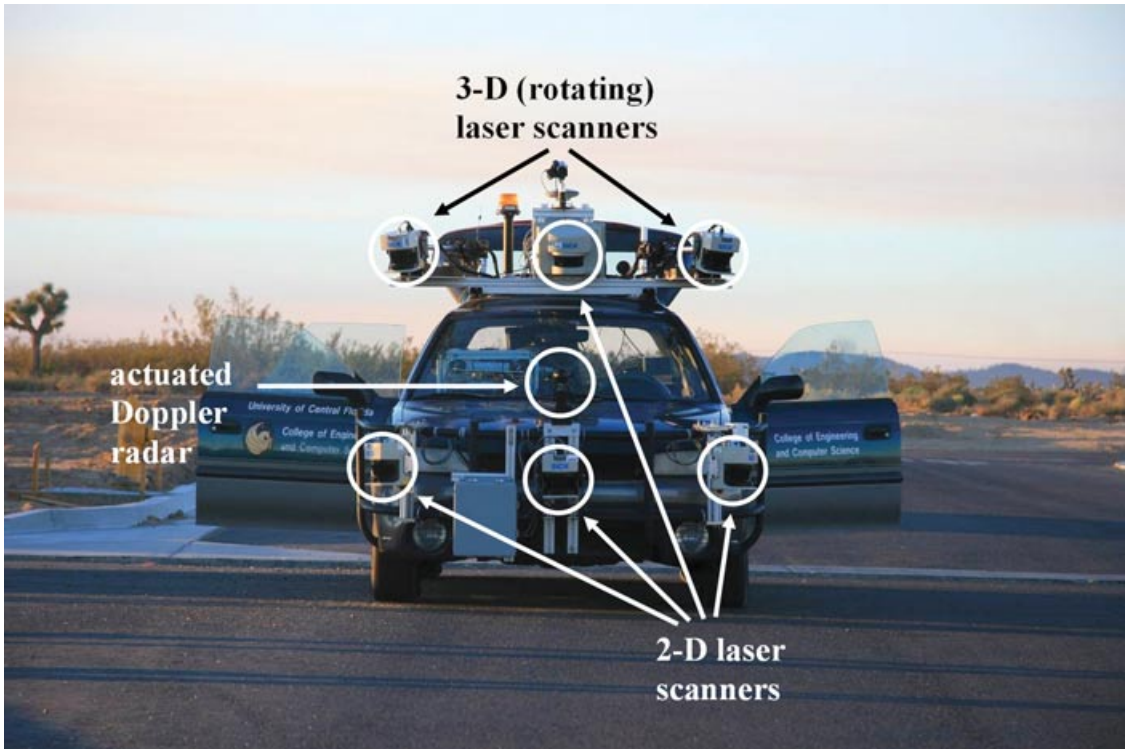

Figure 3. Knight Rider robot before an early-morning test.

particularly valuable in the case in which a single scanner would lose data principally because of looking directly into the sun. TeamUCF saw no benefit to mounting scanners in the "upside down" position that some teams employed in an attempt to reduce the effect of solar glare.

An actuated Doppler radar (Stalker Radar Speed Sensor) mounted at the front of the robot augmented laser scanner data specifically in long-range moving obstacle detection scenarios. This particular sensor employed by TeamUCF provided no effective range or angle information but rather was limited to return intensity and (signed) speed information. This relatively primitive information, however, proved to be a significant advantage in developing the overall system design because it greatly simplified the decision-making logic. Effectively any large object moving sufficiently quickly toward the robot was an obstacle to be avoided.

A Sony HDR-HC3 digital camcorder was mounted on top of the robot and provided a reasonable sensor for lane detection in certain scenarios. Unfortunately, early testing at the NQE showed that solar glare due to early-morning and late-evening operation, coupled with DARPA's decision to use large concrete k-rail barriers as lane markers in many cases, made this video system redundant. For the NQE, the vision system's principal duty was providing a video record of robot performance.

Processing was provided by three core-duo computers (mixed Linux and Windows XP) located in a shock-mounted frame in the passenger's seat. Intelligence functions were performed on one computer and vision functions on a second, and the third computer provided real-time system control including autopilot and navigation functions. Computers communicated over a local Ethernet network, and various processes established connection with one another, in a broadcast/subscribe manner, independent of their actual physical processor location. Communication utilized the Internet communications engine (ICE) framework, which is a simplified derivative of the CORBA architecture.

One principal benefit of the relatively simple system architecture and small number of computers was the relatively low power consumption of

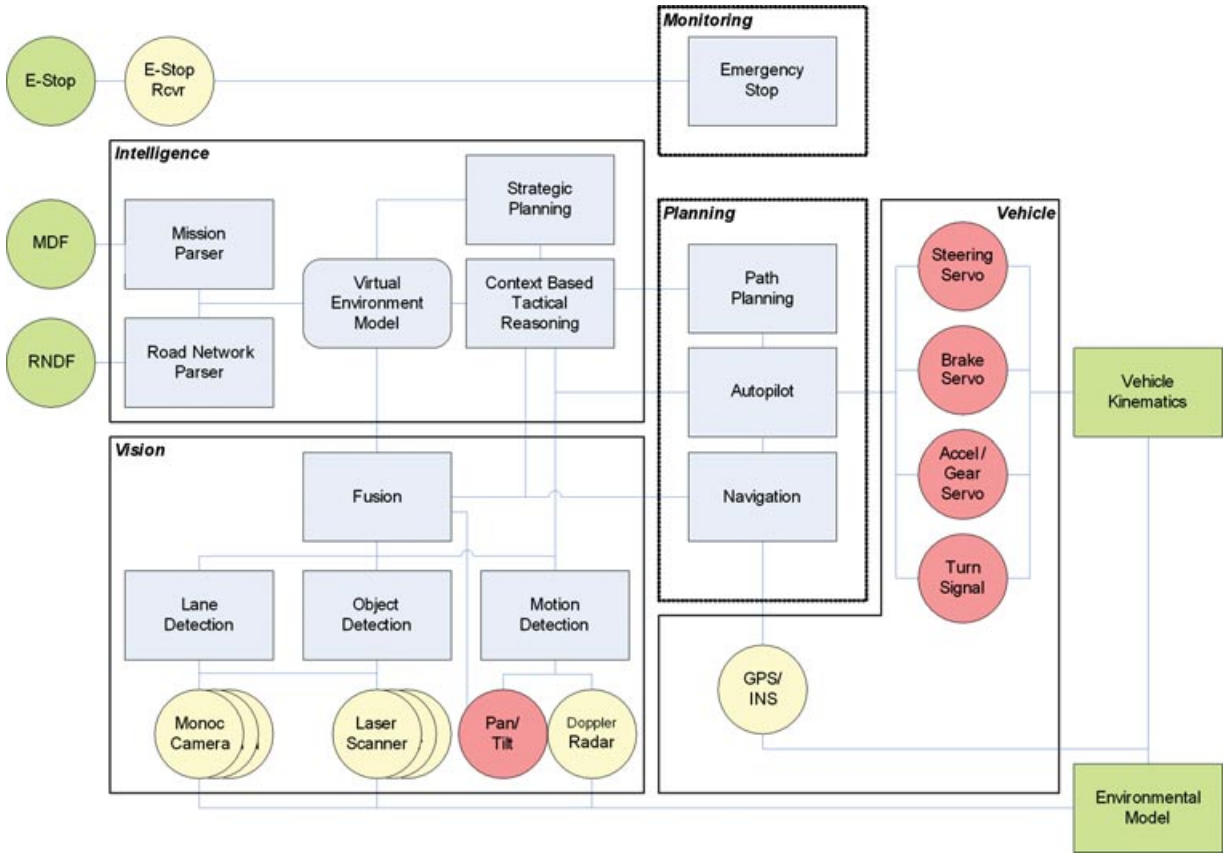

Figure 4. Overall system block diagram.

the robot. Power consumption was ~600 W, which included all sensors and processors. No special alternator was used, and by choice of mounting location, cooling could be provided directly by the robot's air-conditioning system. Computer and sensor power was provided by four deep-cycle marine batteries, which were trickle charged by the alternator. dc–dc converters provided appropriate power levels to various sensors. This system provided stable and clean power and repeatedly demonstrated operation of more than 8 h. Although never required, it was fairly clear that by simply upgrading the alternator, even longer durations could be obtained.

Decomposition of the core software elements is illustrated in Figure 4. For clarity, the detailed interfaces associated with health and status monitoring elements and E-stop are not shown. Clearly visible are overall mission inputs provided by the RNDF (providing an initial seed of the system's environmental model) and the MDF (defining the overall mission objectives in terms of checkpoints and speed constraints). Viewed as a control system, the elements can be considered as follows: 1) intelligence develops a mission as a set of tactical goals to be achieved, 2) path planning efficiently plans a legal and drivable path to meet those tactical goals, 3) the autopilot maintains the robot on path and within performance limits, and 4) proportional-integral-derivative (PID) controllers command various actuators to meet autopilot commands. Although difficult to see in the diagram, feedback effectively consists of four nested loops. The innermost loop consists of PID controller feedback (actuator position, etc.). The next loop consists of navigation information (position, speed, heading, etc.) used by the autopilot to develop control commands to maintain the robot on course. Beyond this is a path planning loop that effectively manages the tactical path based on tactical

goals, bounds, and obstacles. At the outermost level is the overall intelligence loop that monitors whether the robot has met its tactical and strategic objectives. This outer loop is closed through vision as well as navigation.

System operation is straightforward. After the robot boots up and runs an internal self-test, it sits in a wait state ready to accept an RNDF and corresponding MDF. Once files are loaded and successfully processed, the robot remains waiting until released to execute the mission. The detailed mission plan is generated dynamically as the operational environment is discovered. Data logging is performed, allowing mission playback for analysis. Upon mission completion, the robot stops.

2.1. Actuators

The robot's actuator system was composed of four modules, each controlling an existing automotive system. The design of the actuator systems was driven by two overarching principles: to allow for human intervention in any situation and fail-safe operation when no safety driver was present. The ability of a human operator to take full control of the robot at any point is indispensable in extensive testing, and during most testing prior to the NQE, a safety driver was present in the vehicle. To ease the process of relocating and positioning the robot, the actuators were mounted to not interfere with the robot's existing hardware when powered off, allowing a human driver to drive the robot like a normal car. In case the robot is operating fully autonomously with no safety driver, the actuators are constructed to bring the car to a complete stop in the event of a power failure.

The steering controller consisted of a three-phase brushless motor driving a large pulley attached to the existing steering wheel. A six-splined v-belt transferred the torque from the servo motor through a 12:1 mechanical advantage. This small ratio, coupled with the possibility of slip provided by the v-belt, allowed a human safety driver to easily overcome the motor during an emergency situation. The belt design also allowed some compliance to help absorb wheel shock due to potholes and other sudden lateral forces imposed on the front tires. The brushless motor was driven by a 12-A control line from an Elmo 12/60 Harmonica digital servo controller.

Although our design of the steering system allows the belt around the steering wheel to slip, we never encountered any appreciable slip in testing or operation. This was first and foremost a safety feature, allowing a human driver to either overpower a servo motor or slip the belt. Small slippages are compensated by the PID steering controller. For these reasons we did not mount an encoder on the steering column to keep track of the actual steering angle.

The throttle controller consisted of a Bowden cable attached at one end to the original cruise control throttle body linkage. The other end was driven by a magnetic linear motor by Linmot. This type of linear motor was chosen because of its natural ability to release when the dc power was removed. This important safety feature allows the existing throttle return spring to force the throttle closed in the event of an emergency stop or other type of power loss.

The brake controller was a two-part redundant system that allowed control using a linear motor for normal actuation and a separate pneumatic/spring arrangement for emergency stop situations. The linear motor was a larger version of the throttle motor, also by Linmot. The force was transmitted to the brake pedal from behind the firewall using a Bowden cable routed to the actuator located under the passenger seat. The second half of the braking system was used only in emergency situations when either there was a power loss or a disable E-stop had occurred. It consisted of a large spring that, in its natural position, constantly applies force to the brake pedal. During normal operation, a pneumatic cylinder provides a countering force that overcomes this spring and allows the brake to be completely controlled by the linear motor. In the event of an emergency, an electric valve opens to release the pneumatic cylinder, forcing the pedal to be depressed by the spring. An air-release valve controls the rate at which the pneumatic cylinder releases, which in turn controls the stopping distance.

The emergency braking system was specifically designed for the case of a power loss. The electric valve is a three-way solenoid valve that controls the CO_2 flow to and from the pneumatic cylinder that provides a countering force for the mechanical spring. If power is applied to the valve, CO_2 from a reservoir enters the pneumatic cylinder and overcomes this spring so that the brake can be completely controlled by a linear motor. In case of a power failure, the solenoid valve releases the CO_2 from the pneumatic cylinder through its third port, allowing the brake pedal to be depressed by the spring.

The gear shift mechanism utilized yet another linear actuator to provide control over the shifter

position. All of the standard gears (P, N, R, 1, 2, D) could be reached, although normal operation involved only P, R, and D. The existing shift safety interlock was circumvented in this application.

A separate single-board computer running the real-time QNX operating system managed each actuator through either 0–10-V control voltages or, in the case of the steering controller, through a serial port.

2.2. Sensors

Sensor system design was driven by available hardware and proven capabilities, especially from experience gained in the 2005 Grand Challenge. The key requirement of navigation in the Urban Challenge was a safe course traversal in diverse traffic situations. Most scenarios required detection of static or near-static obstacles while the robot was either static or moving slowly (i.e., stop sign scenarios, parking, etc.), but the sensors needed to able to detect and distinguish obstacles at different height levels as well as negative obstacles (potholes). The types of obstacles ranged in size from traffic cones and low curbs to cars, trucks, and major road blockages. As demonstrated by DARPA at the NQE, obstacles were not required to have ground contact with driving lanes. These reflections led TeamUCF to employ laser scanners as the main means of acquiring sensory information. These sensors work very well at moderate range (<50 m) and for the classes of obstacles encountered in an urban environment.

Perhaps the most challenging scenario in the Urban Challenge was the requirement to merge into high-speed (13.5 m/s, ∼30 mph) traffic. Considering car axial acceleration capabilities, safe following considerations, and decision timelines, this required detection ranges of almost 100 m (135 m if a true 10-s gap is to be detected). Range constraints of the laser scanners available to TeamUCF forced the use of a longer range alternative, and TeamUCF employed a Doppler radar to provide the extended range because long-range scenarios involved only high-speed obstacles.

2.2.1. 2-D Laser Scanners

A laser scanner employs emitted laser light and the time-of-flight principle to deduce distances very accurately. Two-dimensional (2-D) laser scanners (LADARs) that use a rotating mirror to provide angular distance measurements in a plane are relatively inexpensive and widely available, especially through the German manufacturer SICK. The biggest disadvantage of those 2-D laser scanners is that they provide distance information in only one scanning plane and hence output only sparse information about the environment. The usable range of distance measurements is 0.5–50 m, with measurement accuracy in the centimeter range.

The disadvantage of just one scanning plane can be partially relieved by mounting several 2-D laser scanners in different orientations. This approach was successfully employed by Stanford's winning robot in the Grand Challenge 2005 (Thrun et al., 2006). TeamUCF decided to place four SICK LMS291-S05 scanners tactically around the car to allow for a near-360-deg field of view (Figure 5). This enabled the detection of static and dynamic obstacles in many possible locations relative to the car. Individually these scanners provide a 180-deg scanning range. They provide complete scans at 70 Hz with 1-deg angular resolution and scans at 35 Hz with 0.5-deg angular resolution. (Both frame rates were used at various times, although NQE testing utilized 35-Hz frame rates.)

For most scenarios the laser scanners in the front of the car provide sufficient sensory information to navigate an urban course. The side scanners in the front are mounted on height levels different from that of the central front scanner. Additionally, they are attached in a slightly rolled position, so that the scanning planes of the three frontal laser scanners overlap in front of the car. These crossing planes focus the attention of the sensors on the area right in front of the car.

Additionally, one scanner is mounted centrally on the roof. With a slight tilt downward, this scanner detects curbs and lane markings approximately 10 m in front of the car (Figure 6).

2.2.2. 3-D Laser Scanners

Despite the design considerations presented, the ability to detect a robust set of obstacles necessitates the use of scanners that perform significantly outside of a single plane (i.e., 3-D laser scanners). Commercial 3-D laser scanner systems are very expensive. TeamUCF chose to emulate 3-D scanning by combining a 2-D laser scanner with a servo motor, such that the scanning plane can be rotated along a chosen rotation axis (e.g., in Surmann, Nuechter, & Hertzberg, 2003, and Wulf & Wagner, 2003). In collaboration with the Mechanical Engineering Department

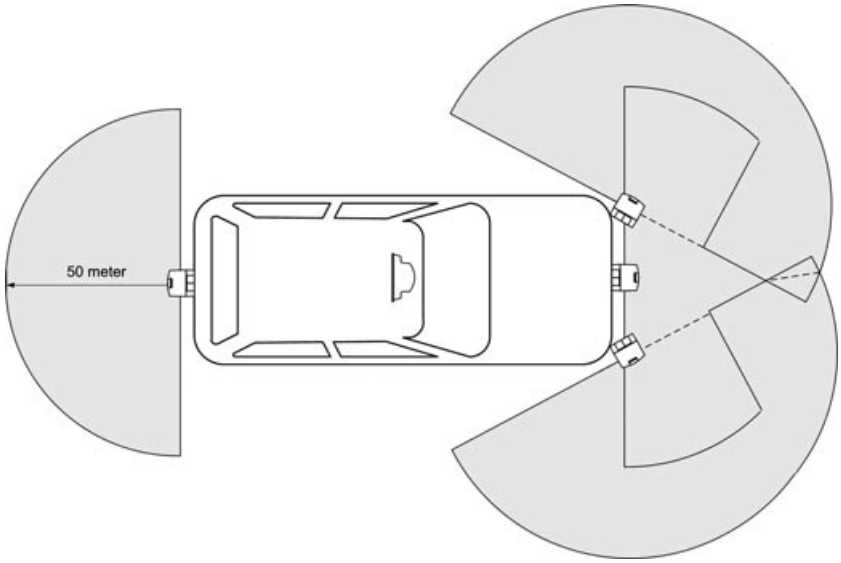

Figure 5. Mounting points of the 2-D laser scanners, plan view.

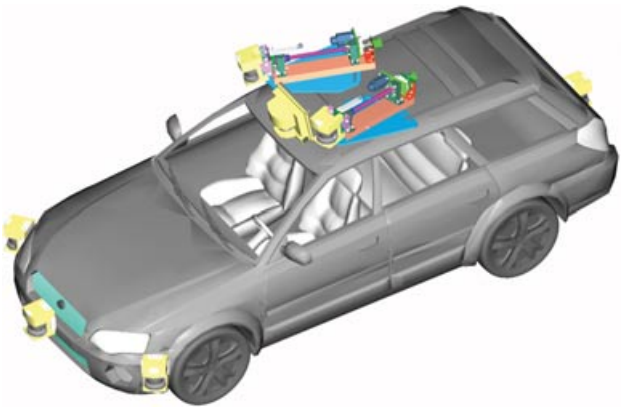

Figure 6. Placement of the laser scanners around the car and on top of the roof.

of the University of Central Florida, we developed an actuated mount that rotates a 2-D scanner to generate 3-D samples, using a single rotational axis and relatively low rotation rate (Figure 7).

In the design of the actuated mount, TeamUCF was guided by three main design paradigms:

- Adjustability: A slotted design allows rapid adjustment of most angles.
- Robustness: Anodized aluminum mount with stainless-steel fasteners for continuous outdoor application. Sealed radial ball bearings resist encroachment of debris and retain lubricant. Teflon plain bearings serve as thrust bearings on either of the front bearing carriers. Rubber bump stops are incorporated to minimize impulse to components should any failure lead to overtravel of the sensor. The cable harness is routed to minimize strain from the repetitive motion and ruggedized with braided sleeving and plastic conduit. All electrical components meet IP65 specification.
- Maintainability: All fasteners, bearings, and electrical parts are off-the-shelf products that are widely available.

TeamUCF considered different motion patterns. The advantage of a continuous 360-deg motion is that the scanners are moved with a constant velocity and hence the interpolation of roll positions is simplified. Unfortunately, challenges are involved when the electrical and data connections have to made through that rotating assembly (e.g., with slip rings). In our testing we achieved a sufficient coverage (e.g., see Figure 8) with a cyclic movement of ± 20 deg. No special connection for the power or data is necessary, and the roll movement of the shaft is closely tracked by an optical encoder. The main reason we chose a cyclic over a continuous movement was the ease of mechanical implementation coupled with a sufficient

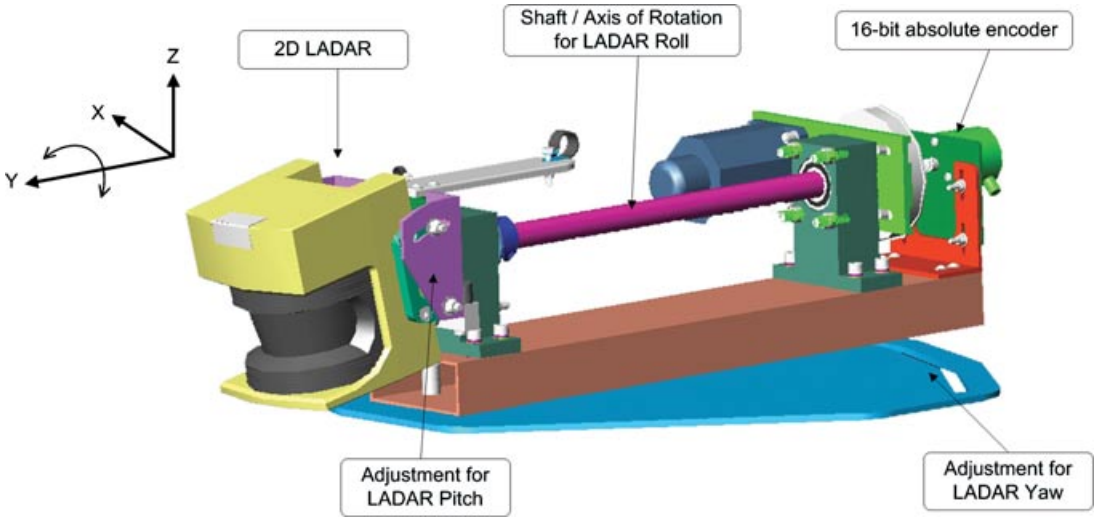

Figure 7. Computer-aided design drawing of the actuated laser scanner mount.

scanning coverage. On the basis of simulations of several configurations and movement patterns, we used two rotating laser scanners that have a yaw angle of ±22 deg and a pitch angle of −11 deg relative to the sensor roll axis. This configuration yields a high scanning point density in front of the robot, where most on-road obstacles are expected.

Each scanner is continuously rotating around the y axis (roll) with a motion radius of ±20 deg. The most current roll position is determined by an absolute 16-bit optical encoder that is directly attached to the rotating shaft. Because each mount has a separate encoder, no roll movement synchronization between the two mounts is necessary.

The point density that is achieved on the ground plane after a 2-s scan is shown in Figure 8. Clearly, the highest point density is achieved in front of the robot. Moderate point densities toward the far frontal left and the far frontal right of the robot favor the detection of robots in intersection and merging scenarios. Notice that the blind spot of one sensor is covered by scan lines of the other laser scanner. Although the point density is low in these areas, the two scanners complement each other in achieving a complete coverage.

2.2.3. Cameras

During early development, TeamUCF used video cameras for lane detection and long-range obstacle recognition. These systems proved to be problematic in testing, being particularly susceptible to variable lighting conditions, ubiquitous shadows, and nonuniform street texture. The basic failure mode in the presence of these conditions was a temporary loss of valid lane data. Promising results in road-marking detection and long-range recognition of oncoming vehicles could not be extended to a robust framework that worked in diverse situations. Further, the performance of the top 2-D laser scanner in detecting lane markings had proved to be at least as robust as vision approaches and could be substantially better in some scenarios. The problems with vision were exacerbated when TeamUCF arrived at the NQE and observed the lighting conditions during expected test windows, the actual quality of road markings, and the extent to which nontraditional road markers (specifically concrete k-rails) were used to designate lane boundaries. TeamUCF made a real-time decision at the NQE and abandoned the idea of using cameras and relied on the laser scanners as the main source of sensor information.

2.2.4. Doppler Radar

The laser scanners employed by TeamUCF were unable to detect obstacles beyond 50 m, but high-speed merge scenarios dictated longer range. To overcome this range limitation, an actuated Doppler radar sensor was mounted in the front of the robot. The

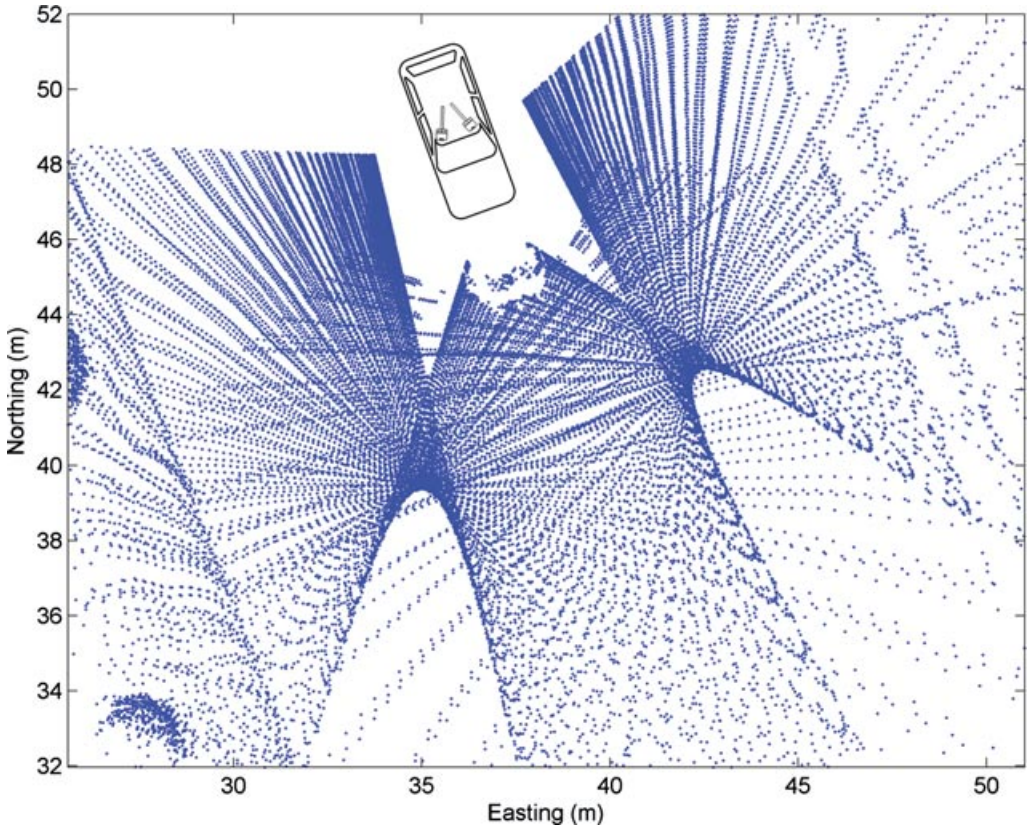

Figure 8. Point density achieved after a 2-s scan.

Stalker Radar Speed Sensor returns the speed of the strongest moving object in its measuring cone (3-dB beamwidth of 12 deg) and has an advertised range of 3 km (ideal for speed traps). In practice, it proved to be a disadvantage to have that extensive range, because the Urban Challenge scenarios effectively limit required range to 100 m. Because the sensor outputs the speed of only the strongest moving target and its direction of movement (incoming or outgoing) but not its distance, a distance-based filtering is not possible. This problem is resolved by simply pitching the radar, so that the maximum detection range is determined by the 3-dB beamwidth (Figure 9). Given the known values of height h, 3-dB beamwidth β, and the desired range d, we can calculate the pitch angle α by

$$\alpha = -\tan^{-1}(h/d) - \beta/2.$$

To account for a diverse range of intersection geometries, the radar was mounted on an outdoor pan-tilt unit PTU-D47 manufactured by Directed Perception (Figure 10). With a maximum yaw speed of 300 deg/s and a maximum pitch speed of 60 deg/s, the radar could view all pockets of the intersection sequentially within a couple of seconds.

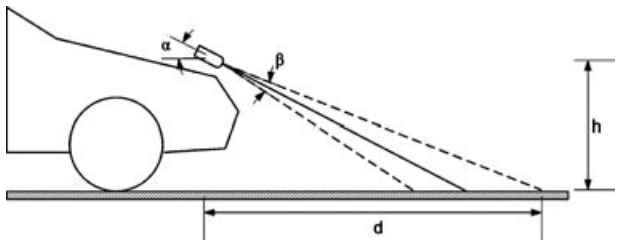

Figure 9. Doppler radar pitched to achieve a desired detection range.

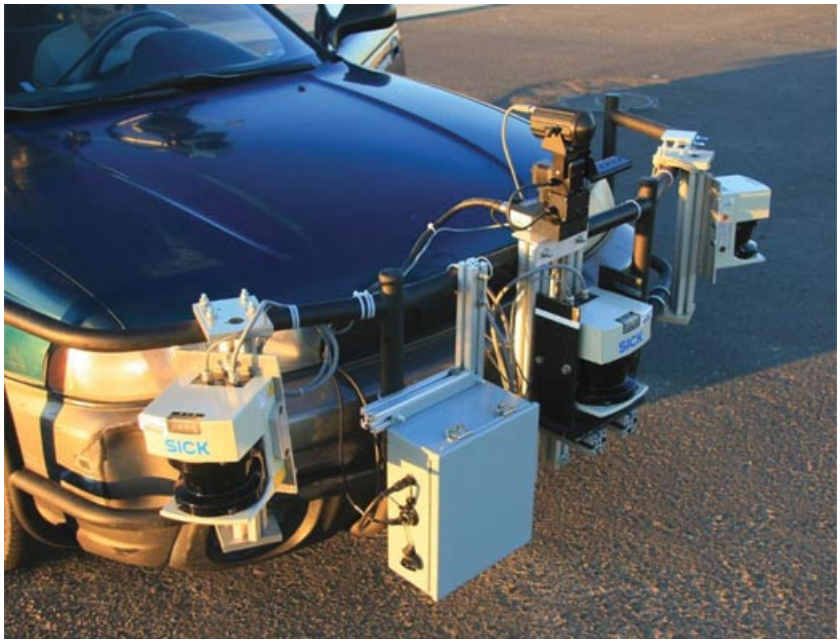

Figure 10. Front rack of Knight Rider with three laser scanners and an actuated Doppler radar.

2.3. GPS/INS

Knight Rider navigation fuses a number of sensors to provide an accurate determination of the current robot state, which includes position, heading, speed, and attitude. Attitude information is used specifically by sensor subsystems to transform sensor relative geometry measurements into a world frame for inclusion in the environmental model. The vehicle's existing antilock braking system (ABS) sensors could be used to provide the current speed of all four wheels, and this information augments states maintained in the Oxford GPS/INS. Differential corrections are provided to the GPS/INS. UCF had investigated dual-antenna performance to augment attitude information, but performance was insufficiently different from that of the single-antenna system now employed to warrant the complexity and idiosyncrasies of such a system. Position accuracy of the operational system was $\ll$10 cm, and angular accuracy was approximately 0.3 deg. GPS/INS data were made available to all processes at a 100-Hz data rate.

3. SOFTWARE ARCHITECTURE

Because of the relatively small development team, little was to be gained by extensive software partitioning. Software was effectively divided into six areas: laser data processing, vision data processing, sensor fusion, intelligence, planning, and control. Each area was owned by one team member (one member owned two areas), who had overall software responsibility, but all team members contributed to all areas of the architecture. A common interface specification allowed seamless data exchange.

3.1. Laser Data Processing and Sensor Fusion

The data from the laser scanners are transmitted over a serial data line with a nonstandard baud rate of 500 kbaud. The serial data are read by a Moxa UC-7110 embedded computer that was modified to support the unusual baud rate. Each embedded computer is capable of receiving data from two laser scanners simultaneously, assigning a time stamp to each scan, and publishing it over a user datagram protocol (UDP) unicast to a central receiver module. Through the middleware infrastructure, the sensor data are made available to the robot's computer network. Several subscriber software modules receive the published data and extract road features and obstacles. Effectively, sensor processing algorithms have access to time-stamped range and intensity data as a function of scan angle, which can be transformed into

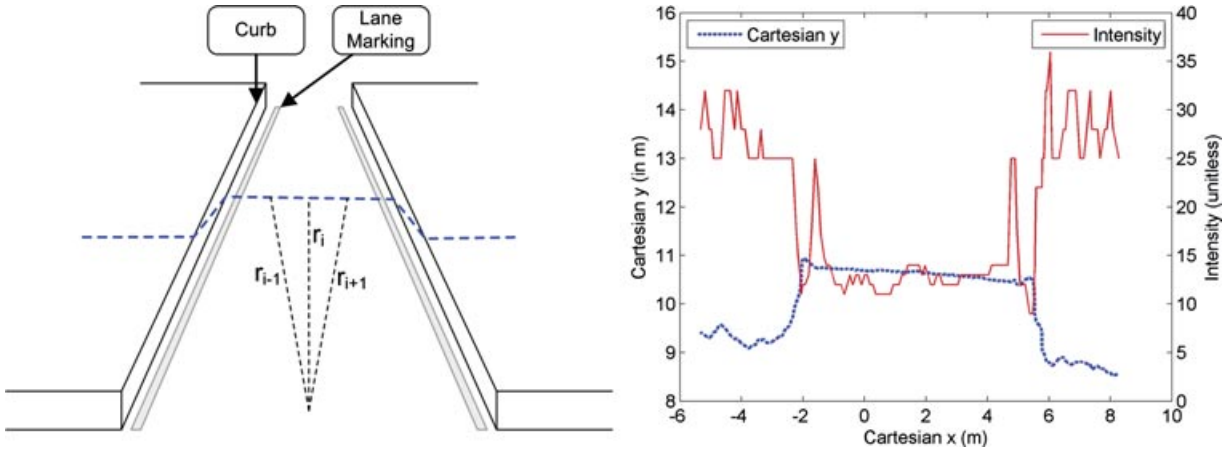

Figure 11. Left: Notional laser scan line. Right: Range (lower curve) and intensity (upper curve) observed on actual road data.

world coordinates through an appropriate transformation involving sensor mounting angles and real-time measurements from the GPS/INS.

3.1.1. Lane and Curb Detection

DARPA provided a collection of way points that were moderately dense (<100-m spacing) but not quite sufficiently dense to rely on way point definitions alone to accurately describe road geometry. Segments with sparse way points were part of the tested road network, and sensory road-following techniques were essential for a safe traversal. Furthermore, even with INS-aided GPS, an intermittent GPS outage and the resulting deteriorating position estimate could have led to inaccuracies in navigation (although TeamUCF never observed this type of GPS failure).

The reflectance of painted road markings is in most cases enhanced by additives such as reflective glass beads. This property facilitates the detection of those markings by a laser scanner. In addition to the range measurement, the SICK laser scanners employed by TeamUCF outputs the intensity of the reflected beam. Range and intensity variations (Figure 11) can be used to define road boundaries.

The road detection strategy employed by TeamUCF was twofold: first, detect the curb discontinuity in the laser range scan (using the top scanner) and then detect the lane marking discontinuity in the intensity scan. Both detections were accomplished in the native polar space of the laser scanner output. By calculating a simplistic range-normalized operator,

$$r_{\text{dot}} = \frac{r_{i+1} - r_{i-1}}{r_{i+1} + r_{i-1}},$$

and thresholding the results, the discontinuities can be easily identified. Data association was relatively simplistic but adequate for the challenges presented. On each measurement, all right-most curb boundaries within $^1/_2$ lane width were associated with the right curb, and all left-most curb boundaries within $^1/_2$ lane width were associated with the left curb.

The detected road/lane boundaries are then tracked by a second-order Kalman filter, which ensures that broken curbs or broken lane markings do not seriously impact the estimated boundary points. Because the RNDF input format guarantees that way points, when present, are accurate and that lane widths, when present, reasonably represent the lane, the only point of interest for the environmental model is the world coordinate of the center point for the current lane of travel. This point is used by the environmental model if the spacing between known points is larger than a threshold and essentially became an additional, lower confidence, waypoint. (TeamUCF sought a way point spacing of 10 m.)

3.1.2. Obstacle Detection with 2-D Laser Scanners

To detect and extract obstacles, laser scanner points were transformed into world coordinates and fed into a probabilistic occupancy grid originally developed

by Moravec (1988) and excellently treated in Thrun, Burgard, and Fox (2005). The occupancy grid is probabilistic in the sense that it represents the map as a field of random variables, arranged in an evenly spaced grid. Each grid cell is either occupied or not, and hence the random variable is binary. An occupancy grid-mapping algorithm implements an approximate posterior estimation of those random variables. TeamUCF utilized a grid cell size of 0.5×0.5 m. If a scan point is within the grid cell, the cell counter is incremented and compared to an occupancy threshold. The line between the laser's origin and the scan point is traced, and the counter of traversed grid cells is decremented.

TeamUCF modified the standard concept of an occupancy grid to specifically suit mapping of an urban driving environment. In particular,

- TeamUCF utilized a 2-D occupancy grid. It is not important to know at which particular height an object resides but that it exists in a height bracket above (or below) the road level that poses a danger for the robot. Points outside this band (either too high or too low) are discounted from consideration in the grid.
- TeamUCF used a dynamic moving map in order to minimize memory and computation expense. Intelligence and planning systems are concerned about detailed obstacles only within the vicinity of the robot. (Obstacles outside this range are likely to change.) TeamUCF used a robot-centered 120×120 m occupancy grid that was moved whenever the robot moved 10 m. This ensured that all obstacles in at least a radius of 50 m around the car (effectively the maximum range of the laser scanners) were mapped.
- TeamUCF required data for each grid cell to be refreshed repeatedly or lost over time. This reduced the impact of potentially outdated data from cells not effectively resampled by the laser scanner within 2 s (due to robot motion or more likely orientation). This "fading" of occupied cells was implemented with time stamps.
- TeamUCF used a dynamically generated lane mask to further eliminate obstacles outside the road network. The lane mask was maintained by the environmental model and reflected the best estimate of the road network. Cells sufficiently far from the road network were simply ignored. The mask was communicated as a collection of potentially overlapping polygons. The vehicle relies on accurate GPS data, and masks were selected based on nominal GPS drift of <1 m. Masks were "realigned" only to the extent that roadway control points (initially RNDF points) were updated as the vehicle traversed the path. In fact, this roadway update was a far more significant source of path changes than GPS drift, often adjusting the roadway by many meters.

The point transformation and occupancy grid mapping was executed separately for each of the four statically mounted laser scanners at a rate of 20 Hz. At the end of each iteration, the resulting occupied cells of all four grids were published to the sensor fusion module.

3.1.3. Obstacle Detection with Rotating 3-D Laser Scanners

An important observation from the process of curb detection is the apparent smoothness of the road surface. In fact, all obstacles that have to be avoided by the robot distinguish themselves as a discontinuity in respect to their spatial surroundings. The simplified operator used in the curb detection process is insufficient in cases in which the laser beam hits the scanned surface at an extremely acute angle, because larger changes in the measured ranges can be expected even if the observed surface is smooth.

One key advantage of using a rotating 2-D laser scanner to emulate a 3-D scanner is the preservation of spatially continuous scan lines. That is, each pair of adjacent scan points in a given scan line is in most cases spatially close in the observed environment, so that an evaluation of surface smoothness along the scan line is possible. Ideally, an operator on the scanning data returns gradient changes independently of incidence angle of the laser beam and scanning location relative to the environmental feature.

As elaborated in Adams (2001), the change in gradient from two scan points A, B to the new scan point

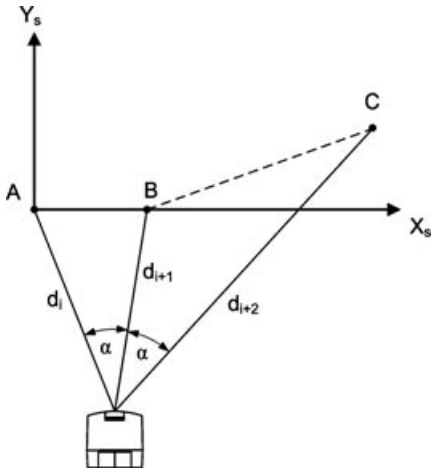

Figure 12. Three sequential scan points A, B, and C.

C can be described by

$$\Delta \left| \frac{dy_s}{dx_s} \right|_{x_s=B} = \frac{(d_i d_{i+1} + d_{i+1} d_{i+2} - 2 d_i d_{i+2} \cos\alpha) \sin\alpha}{d_{i+1}^2 - d_i d_{i+1} \cos\alpha - d_{i+1} d_{i+2} \cos\alpha + d_i d_{i+2} \cos(2\alpha)},$$

where α denotes the angular resolution of the laser scanner and d_i, d_{i+1}, and d_{i+2} the observed range measurements to points A, B, and C, respectively. The key here is the definition of a local coordinate system (x_s, y_s) in the sensor space once the first two points are scanned, where the x_s axis is joining the two points A and B, as illustrated in Figure 12. The computed gradient is then the gradient in any chosen coordinate system, no matter from which side the environmental feature is scanned.

Thresholding the resulting gradient changes in a given scan line yields the points that describe the sought-for spatial discontinuities in the environment. Appropriately choosing the threshold allows detection of objects as diverse as road curbs and cars. The resulting points are transformed into world coordinates similar to the transformation for the static laser scanners, with the added degree of freedom for the roll motion. An interpolation of the roll angles for each point of a scan line accounts for the continuous movement of the scanners. An example point cloud before and after the described processing is shown in Figure 13. All the remaining 3-D scan points were inserted into a separate occupancy grid, as was described for the 2-D scanners.

The scan line gradient processing, transformation of the remaining points to world coordinates, and occupancy grid mapping are performed data driven at the full laser scan rate of approximately 35 Hz.

3.2. Sensor Fusion

The sensor fusion module receives five occupancy grids from the laser processing modules, four from the static 2-D scanners and one from the 3-D laser scanners. Each of these grids represents a probabilistic "best guess" about obstacle locations from the particular sensor's point of view, suggesting to merge the grids disjunctively. The resulting disjunctive occupancy grid contains all known obstacle cells in a perimeter around the robot (Figure 14).

From experiments it became clear that the occupancy grids from the 2-D scanners were more likely to contain false positives due to unusual road geometry or rapid elevation changes in the environment. In contrast, obstacle cells extracted by the 3-D laser scanners proved to be more reliable indicators of real obstacles. To avoid deadlocks due to false-positive obstacles, obstacle cells from the 2-D laser scanners that are nonexistent in the grid of the 3-D scanners were deleted after a predefined deadlock time. This mechanism is essential and formed the basis of our approach to driving on hilly terrain.

Separate obstacles were extracted by a connected component analysis, and subsequently their polygonal outline and centroid were determined. The grid-based representation of the world allows for a natural quantization of the coordinates of polygon vertices. Based on position and velocity from the previous sensor fusion iteration, expected obstacle locations were extrapolated and current obstacles were assigned consistent IDs based on minimum Euclidean distance to the expected locations. If no correspondences for previous objects could be found in a certain radius, their ID was deleted. Conversely, if new objects appeared that could not be matched, a novel unique ID was assigned to them.

Accurate tracking of obstacle velocity cannot be achieved on the grid level due to the coarseness of the occupancy grid. Fortunately, there is a good chance that the object is directly visible in at least one of the laser scanners, whose distance measurements are accurate to within 4 cm. By transforming the obstacle centroid from world coordinates to laser

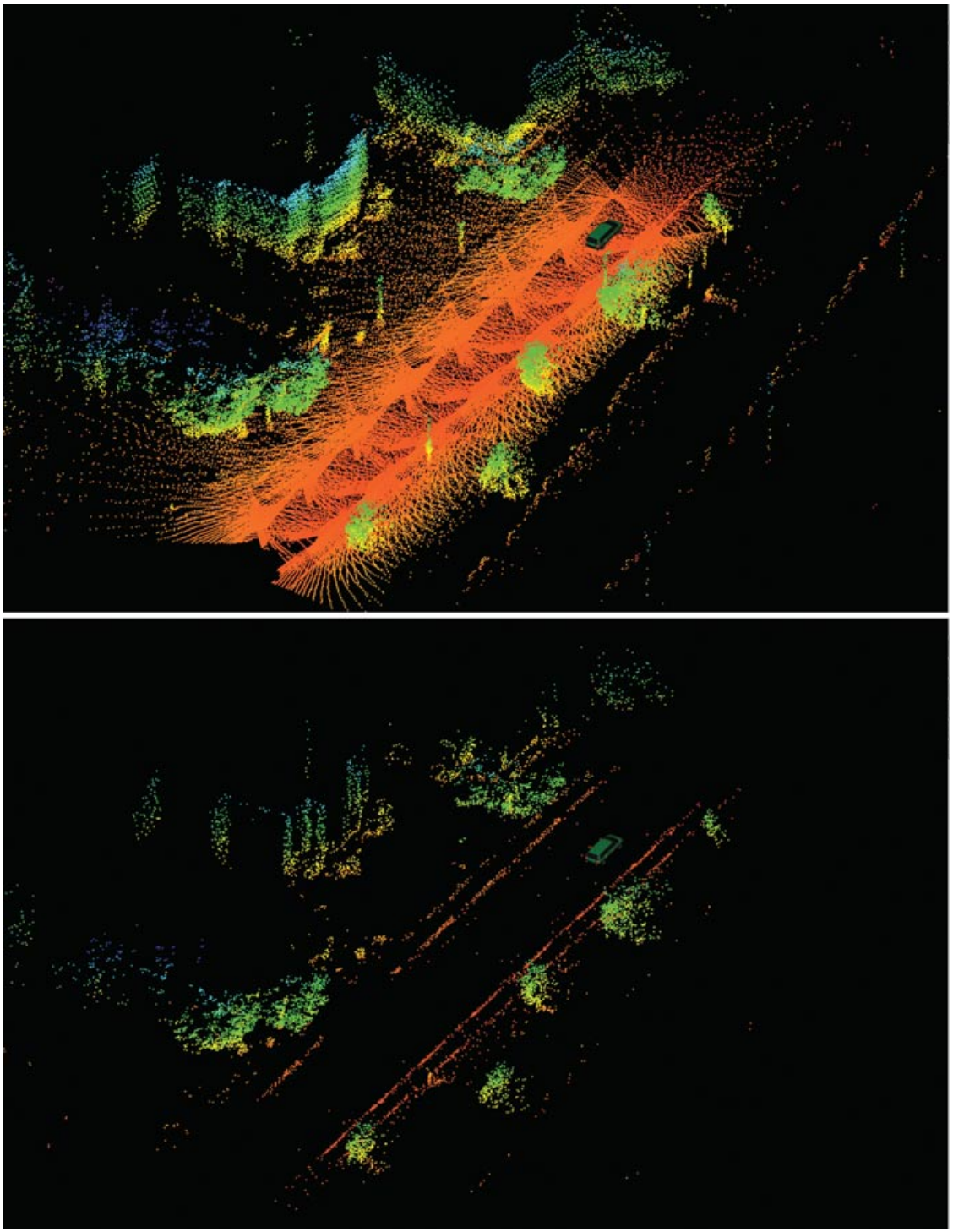

Figure 13. Top: Typical 3-D point cloud. Bottom: Processed points prior to occupancy extraction.

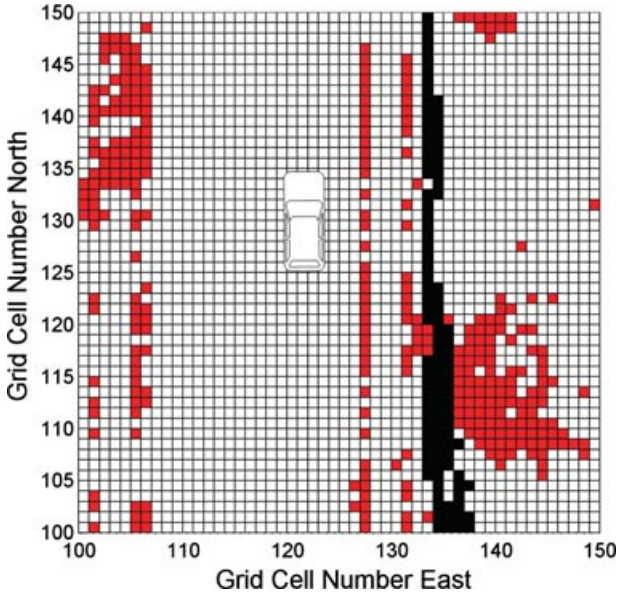

Figure 14. Snapshot of an occupancy grid fragment, 2-D (darker) and 3-D (lighter) cells.

coordinates, it is determined whether it is within the line of sight of any of the scanners. If a corresponding scan point can be found at the expected obstacle range, its transformed world coordinates can be used to track velocity through sensor fusion iterations. The resulting velocity assigned to a specific object ID is stabilized by an exponential moving average filter. If an obstacle's velocity is below 1 m/s, it is assigned a static flag and an associated time it has been observed not to be moving.

Another task for the sensor fusion module was to estimate the geometry of the travel lane by storing and extrapolating the lane center points extracted by the curb/lane detection. The center points received within the last 20 m of travel were approximated by a second-order least-squares fit parameterized to work directly on UTM world coordinates. Specified points in front of the car at distances of 5, 10, 15, and 20 m along the second-order curve were extracted.

All the sensor fusion processing is performed at a rate of 5 Hz. A full list of known obstacles with ID, velocity, static flag, and static time, as well as a list of lane center points in front of the car, is published to the artificial intelligence (AI).

The calibration of the positions and static angles of the laser scanners with respect to the car coordinate frame was performed in testing prior to arrival at the NQE. The calibration approach was practical. We scanned previously known features (such as a corner of building) that showed up in multiple laser scanners and found yaw/pitch/roll and translation with respect to the car with a 3-D iterative closest point (ICP) algorithm. The changing roll angle of the rotating scanners was very closely tracked by the optical encoders mounted on the moving shaft. Timestamping the car state information, received laser data and encoder data, allowed interpolation of car state and laser state for each scanning point. This approach worked very reliably, although it relies heavily on the accuracy of the GPS/INS system.

3.3. AI: Intelligence

The AI module was responsible for the high-level planning and tactical-level decision making for Knight Rider. In designing the AI module, heavy emphasis was placed on existing research in driver modeling approaches. Development of driver models is integral to the quest for a better understanding of how humans drive, which in turn supports effective in-vehicle interfaces, better collision warning and avoidance technologies, and improved driver-related human factors. Driver models used for simulating traffic in immersive driving simulators are particularly appealing because of the requirement for realistic-looking behaviors that extend all the way to faithfully reproducing motion trajectories. The AI module used in Knight Rider was based on a driver model derived from prior work in driving simulation (Cremer, Kearney, & Papelis, 1995; Papelis & Ahmad, 2001), but with several extensions and enhancements to address incomplete awareness of the driving environment and rules and requirements of the competition.

Competition-specific issues aside, the driver model was structured according to Michon's three-level hierarchy (Michon, 1985) that breaks the driving task into strategic, tactical, and operational levels. The strategic level is concerned with high-level goals such as navigation. The strategic level mapped these goals into a series of subgoals, which remain unchanged unless affected by external factors. The tactical level was responsible for generating sequential tasks to implement a given subgoal. The operational level was responsible for low-level guidance of the robot. The three-level hierarchy provided an effective cognitive model and is consistent with the

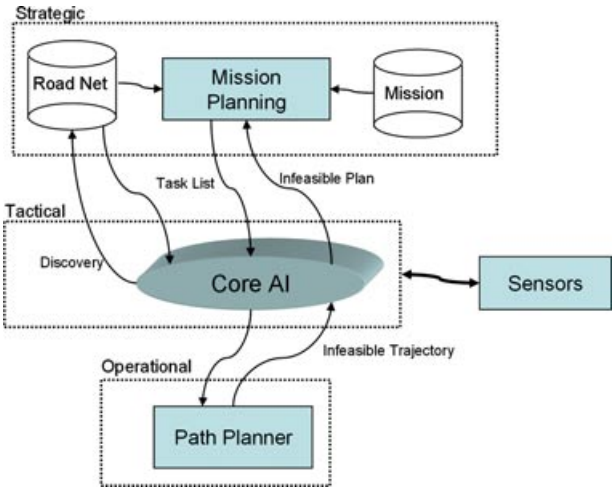

Figure 15. Behavioral model block diagram.

view that driving is a compromise between achieving goals and addressing ongoing constraints (Boer & Hoedemaeker, 1998). Through a temporal process of selection of alternatives, the driver model pursues goals in a top-down fashion, starting at the strategic and ending at the operational. Constraints flow the opposite way, starting at either a tactical or the operational level and reaching the strategic level, which in turn adapts accordingly.

3.3.1. AI Architecture

Figure 15 depicts the decomposition of the driver model into the three behavioral levels and associated flow of tasks, from top to bottom, and constraints, from bottom to top. The road network information was read from the RNDF and converted into an indexed data structure that better supports robot navigation. This step also created several needed associations that are not explicitly provided in the RNDF, for example, lane adjacency and direction information. The MDF was read and used to plan a mission, which in turn was used by the mission planning logic to create a list of tasks. These tasks were implemented within the core AI module, which used sensor data and a priori knowledge to execute the specific tasks. Low-level motion requirements in the form of a series of geometrical points to drive along was passed to the path planner, which interacted with the low-level control mechanism to ensure proper robot motion.

3.3.2. Strategic Level

For the strategic level, establishment of the goals and the associated optimization functions was done by interpreting the competition rules. Materials provided before the competition provided specific operational boundaries, but no quantitative scoring information was given. As a result, the strategic level is focused almost exclusively on route/mission planning and dynamic replanning upon discovery of road blockages. The output of mission planning was a list of tasks, each of which corresponds to a tactical operation, such as driving and parking.

Early performance testing using the hardware employed in the robot indicated that a straightforward implementation of Dijkstra's algorithm performed almost instantaneously on graphs with hundreds of nodes. At that point, work was underway on generating a graph from an RNDF, but even under the worst-case assumptions, Dijkstra's $O(N^2)$ algorithm performed adequately, so the decision was made to utilize this approach for determining the route from one checkpoint to the next. A simple algorithm was designed that starts by finding the best route from the current position of the robot to the first checkpoint, then augmenting that route by the best route from the first checkpoint to the second, and continuing until all checkpoints have been exhausted. To successfully utilize this approach, two specific issues had to be addressed: first, development of an algorithm that would convert an RNDF into a graph amenable to min-path search, and second, developing a mapping between a route and a series of tasks that could be delivered to the tactical level for execution.

3.3.3. Graph Generation and Task Mapping

The traditional min-path algorithm finds an optimal route between two nodes on a graph. Optimality is defined in terms of the route cost, which is the cumulative sum of the cost of traversing each node and edge of a given route. In generating a graph from an RNDF, it is important to capture all navigation possibilities inherent in the topology as well as to capture a rational cost function that can be used to compute node traversal cost. The graph generation algorithm developed to address these issues involves two phases. The first phase was responsible for creating the graph nodes, and the second phase was responsible for generating appropriate edges. The algorithm

built a graph at the beginning of the mission and rebuilt the graph as needed during the mission.

To generate nodes, the algorithm considered all way points and included as unique nodes any way points that were

- an exit originating on a lane
- an exit originating on a zone
- an exit target, on either a lane or a zone
- the first or last way point of a lane
- a parking spot that was a checkpoint of the current mission

Edges were generated from each node under the following conditions:

- If the node represented an exit from a segment or from a zone, edges were created to all destination nodes.
- If the node was not an exit on a segment, a single edge was created to the nearest node located downstream on the same lane.
- If the node represented an entry zone way point, edges were created to all parking spot nodes in the same zone and to all exits in the zone.
- If the node represented a parking spot, edges were created to all nodes representing zone exits.
- To represent lane changes, edges were added between nodes on the same road that were on different lanes and downstream from each other. Such edges were added only when the RNDF topology allowed a lane change, that is, when a dashed white lane separates the lanes.
- For roads with two lanes of opposite direction, and for which a corridor allowing a U-turn did not exit, an edge was added between the last node of a lane and the first node on the adjacent lane. This edge allowed the routing algorithm to schedule U-turns at the dead end of two-lane roads.

Figure 16 gives an example of the graph generation process.

Once edge generation was completed, a classifier was used to assign each edge to a tactical-level behavior that could handle the narrow problem of navigating the robot so that it traversed from one node of the graph to the next. Associated with each tactical-level behavior was a cost function that produced an estimate of the cost associated with navigating this edge. Once costs were associated with the edges, the min-path algorithm was applied to generate a linked list of edges. The list defined the anticipated series of tactical-level behaviors that were invoked during the mission. A final step allowed reaching checkpoints by performing midroad U-turns. During this step, the min-path search was performed four times, once with no U-turns, once with a U-turn from the current location and straight arrival to the checkpoint, once with a straight departure but arrival to the checkpoint after a U-turn, and once starting and ending with a U-turn. The minimum cost path was selected.

It is important to note that consideration of midroad U-turns was incorporated in the algorithm even though such U-turns were considered illegal. The rationale for this decision was simple. Without knowing the relative cost of time performance versus illegal moves, it was unclear whether the penalty of the illegal U-turn would be offset by the time gain. Incorporating the midroad U-turns in the algorithm provided more options than not having this capability at all.

Modifications to the weight function can be used to bias the behavior of the robot. For example, the cost of U-turns affects the choice between driving down a dead-end road and performing a U-turn versus driving around a larger loop that involves no U-turn. During testing and during the competition, experience and improved rule understanding yielded several calibrations of the weight functions that proved to be critical in the success of TeamUCF during the NQE.

One example of such a calibration was elimination of midroad U-turns. During the NQE, it became clear that the time it took to complete any one of the courses had little weight when compared to safely finishing the course. The decision was made to eliminate midroad U-turns, which was achieved by modifying the weight function so that it assigned a very large cost to such a maneuver.

3.3.4. Tactical Level

The tactical level was focused on implementing the list of tasks produced by the strategic level. The tactical level was also responsible for road discovery. Road discovery is the process by which existing lanes are augmented with sensor data that provide a fuller centerline description than was originally available. To support road discovery, a confidence value was

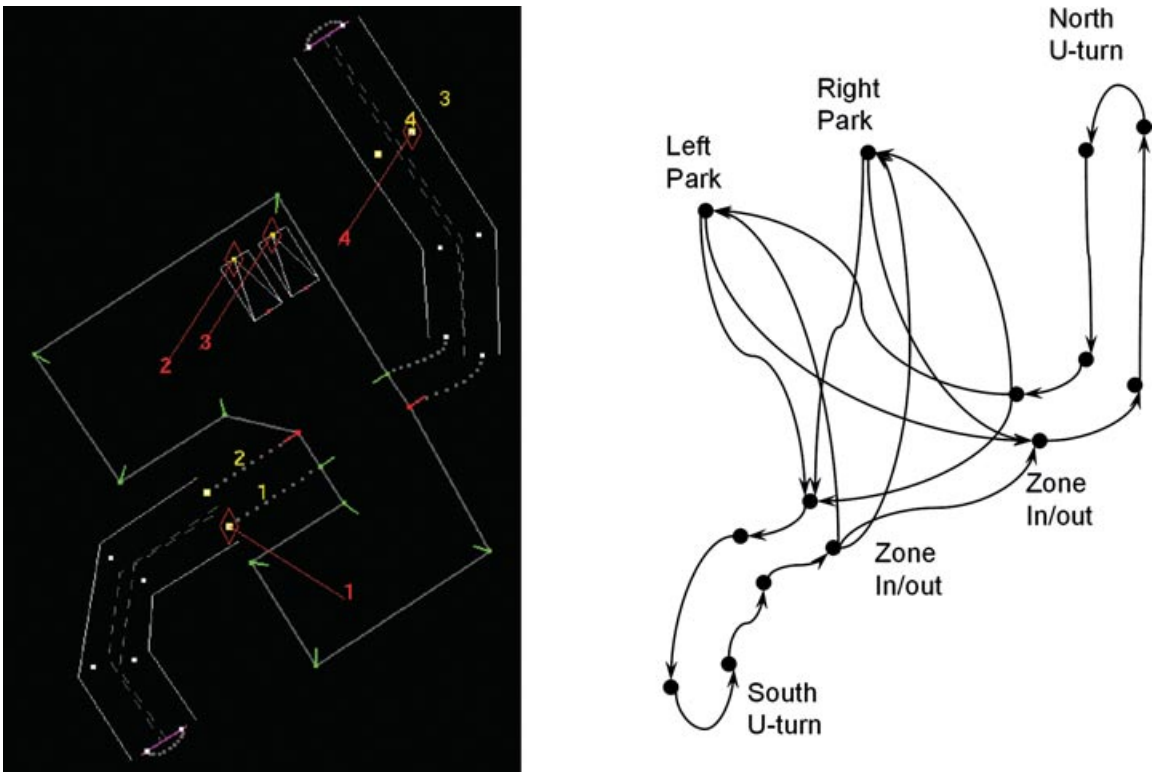

Figure 16. Example of route generation.

associated with each way point. Initially, all known way points receive a confidence of 0.9, to indicate full knowledge of the (x, y) coordinate but incomplete knowledge of the heading. As the robot traveled over a way point, the heading was updated and the confidence reached the maximum value of 1.0. At the same time, when way point density was below a threshold, guidance was provided by tracking the lane centerline ahead. This information was used to add way points into a lane, but with a lower confidence than the points specified in the RNDF, which were considered ground truth. The confidence of new points was passed from the sensor module. This process allowed the incremental increase in the confidence of newly inserted way points when repeated traversals over the same road segment occurred.

The tactical-level implementation framework was a hybrid model that borrowed elements of context-based reasoning and state machines. Context-based reasoning allows a functional decomposition of the problem space into subspaces that are easier to handle. Each context is responsible for observing the current situation and "offering" to solve the problem at hand. Even though the context-based formulation does not directly address concurrency, it does allow nonorthogonal activities to exist in multiple contexts, and in practice this is simply implemented by cleverly designing reusable behavior objects. A fixed-priority assignment was used to pick the context that takes control, if more than one context was willing to do so.

Even though the context-based approach has several advantages, it also presents some disadvantages. In particular, it does not lend itself to implementing procedural, step-by-step actions that are typically encountered in driving. A state machine approach is much better suited to this type of behavior. To facilitate modularity, a hierarchical state machine (HSM) model was used within each context to implement the appropriate behavior.

Figure 17 depicts the hybrid model of a context. The enable function is used to indicate whether the

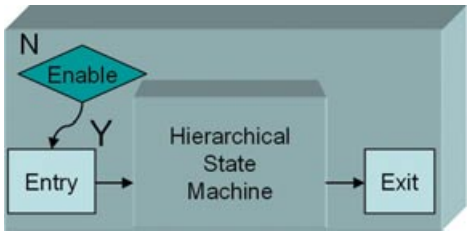

Figure 17. Internal structure of a context.

context is willing to handle the situation. If the result is affirmative, the entry function executes to provide consistent initialization activities. The HSM then takes over while the context is active, and upon exit, a termination function provides a consistent point that performs context-specific cleanup activities.

The full execution semantics are illustrated in Figure 18. At start, the enable function of each context was executed and the first one that returned true activated the context and the associated entry function. The HSM code then executed periodically. If a higher priority context took over, the exit function was called and the selection process repeated.

A common problem associated with context-based behavioral modeling is maintaining continuity

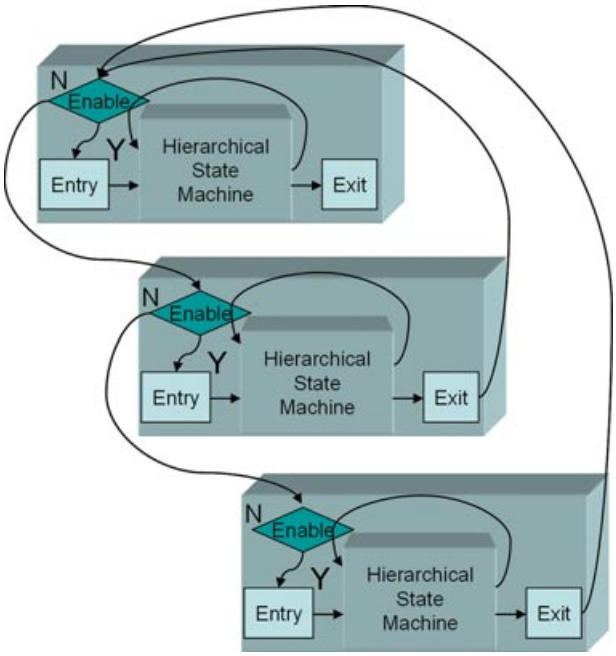

Figure 18. Context execution.

of behaviors during context changes. The localization achieved by using contexts is inherently incompatible with the need to maintain smooth transitions during context changes. For example, consider a context responsible for driving along a lane on a road with the speed limit set to 30 mph (48 km/h). Let us further assume that the road leads into a stop sign, which is handled by a different context. The context responsible for driving is not aware of which context follows; that would violate the locality inherent in the framework. As a result, the driving context maintains the maximum speed and depending on where the transition occurs, the context dealing with the stop sign can receive control so near the threshold that stopping is not physically possible. To address this problem, each context was structured so that it composed the control inputs passed to the lower levels through an overloaded method that accumulated trajectory way points using the best available information at any time. It was thus possible for a context to recursively call the trajectory augmentation routine of subsequent contexts without explicit knowledge of which context followed. By maintaining a minimum length of trajectory specification, the path planner could anticipate speed as well as direction changes and plan accordingly. Using an overloaded method maintained the context independence while satisfying the need to plan ahead.

This loop executed in periodic fashion in soft real-time mode. The tactical-level thread was the primary thread within the AI process, with the strategic and operational levels implemented as separate threads that executed when triggered by the tactical level. In the actual robot, the execution rate was set to 10 Hz, leaving 100 ms per iteration. Use of asynchronous threads facilitated development and decoupled the tactical control of the robot from the variable execution performance associated with the other threads.

Figure 19 illustrates the specific context design used in the Knight Rider. The prioritization order was designed to arbitrate between overlapping domains. For example, handling an intersection with stop signs had higher precedence than a regular intersection. The lowest priority context (default) served several purposes. During development, it acted as a self-check mechanism that pointed out gaps in the system. During autonomous navigation, it served as the central place in which a last effort could be pursued to handle an unexpected situation.

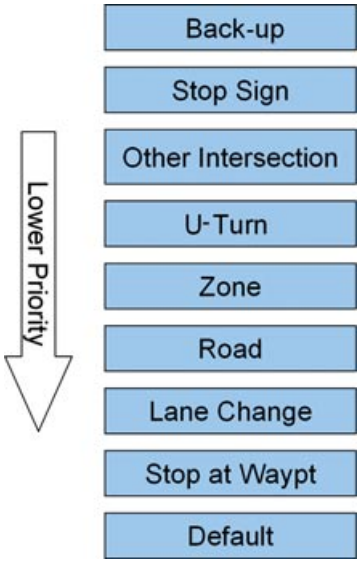

Figure 19. Context design.

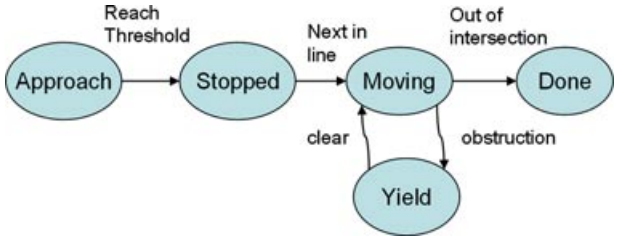

Figure 20. HSM for stop sign.

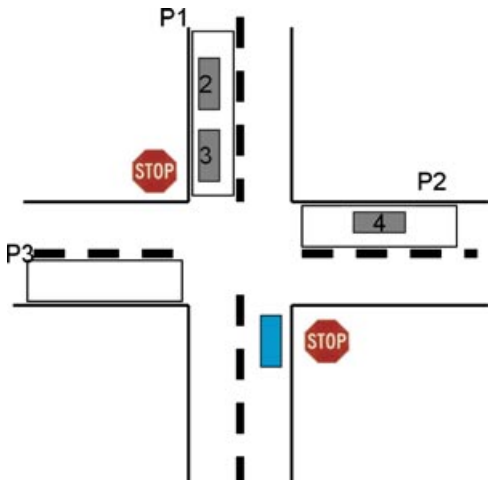

Figure 21. Illustration of pockets.

3.3.4.1. Back-Up

The intention of this context was to drive the robot in reverse when doing so would allow meeting a checkpoint located behind the robot. This context did not directly map to an activity produced by the route planner but as the highest priority, context had the opportunity of checking for this situation. The context consisted of a single state that attempted to back up only when the next checkpoint was located within three robot lengths and there were no obstacles in the way.

3.3.4.2. Stop Sign

This context was activated when the robot must cross an intersection from a lane that was controlled by a stop sign. The structure of this context is straightforward, as illustrated in Figure 20. Note that substates used to implement timeouts are not shown.

Upon activation, the robot approached the threshold and came to a stop. Information about the intersection geometry was utilized to create a set of *pockets*, each representing other lanes into the same intersection. Pockets were classified as *peer* or *high* priority. Peer pockets were ones controlled by a stop sign, whereas high pockets had no signage. The operation of the stopped state differs between the cases when all other pockets are peer versus having at least one high-priority pocket, but in both cases, the robot remained in the threshold as long as an object was inside the intersection. Figure 21 illustrates an example intersection. The robot is approaching from the south. In this case, pocket P1 is peer and pockets P2 and P3 are high priority.

When all pockets were peer, the presence and velocity of objects in the pockets was used to determine right of way. Empty pockets or pockets with moving objects were ignored; pockets with a stopped object were assigned right of way. Once an object in a right-of-way pocket moved, the pocket was eliminated. This ensured that the robot waited only for the lead object when multiple objects were queued on a peer pocket. This logic brings up an important observation. The Knight Rider robot assumed that other vehicles (robots or cars) would behave according to the rules of the road, and only as a final resort (object all the way in the intersection) did the Knight Rider stop. During testing it became apparent

just how many subtle cues human operators obtain from other drivers' behavior and from the drivers themselves, cues that were not available to the Knight Rider.

When high-priority pockets were present, the stopped state did not transition as long as other objects in these pockets were in conflict. The velocity of the oncoming traffic was used along with their distance from the respective thresholds to determine whether a conflict existed. When conflicts and right-of-way rules had been resolved, the robot transitioned into the moving state, which lasted while it was inside the intersection. Traffic entering the intersection forced a transition to the yield state, during which the robot stopped. Upon clearing or after a time-out period, the robot proceeded.

Time-outs were used in all waiting states to prevent live lock, which could be caused by other robots that intentionally or unintentionally violated the rules or by phantom objects due to sensor artifacts. Such time-outs were set at such a high value that they would never interfere with typical interactions. Further, TeamUCF made a calculated decision to prevent initiation of a passing maneuver when near an intersection, but once a pass maneuver was initiated it would be completed, even if that meant passing while approaching a stop sign.

3.3.4.3. *Intersection*

This context was responsible for controlling the robot through intersections for which there was no stop sign (Figure 22). The most obvious situation is a left turn that crosses opposite-lane traffic. Because California driving rules dictate a full stop before crossing a yellow line, the design was similar to the stop context, but with two key differences. Because there was no intersection area, once a go decision was made the robot proceeded without monitoring for side obstacles. (It was assumed that those moving obstacles would stop.) In addition, there was no consideration for incident lanes controlled by stop signs as they had lower priority.

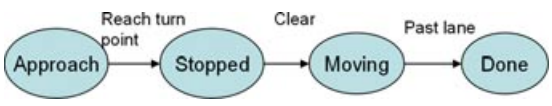

Figure 22. HSM for intersection context.

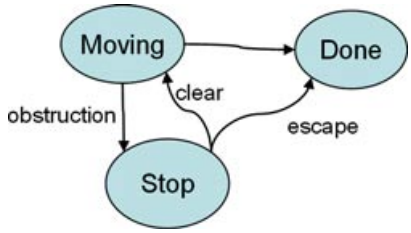

Figure 23. HSM for U-turn.

3.3.4.4. *U-Turn*

This context was responsible for implementing U-turns. The context activated in two situations. The first case was planned when the next activity in the route list explicitly called for a U-turn. The second case was when a blocked road was encountered. The only difference between these two cases was that the latter case also triggered a rerouting operation at the strategic level, which generated a route starting at the lane that was the destination of the U-turn. In both cases, the context terminated upon completion of the maneuver.

To ensure that the reroute operation would not create a route that traverses the same blocked road, the edge representing the blocked road was tagged with a marker indicating the location of the block. According to competition rules, blockages were not persistent and the block marker was removed after the robot crossed a corridor, in effect forgetting the blockage

Despite the relatively complex sequence of operations necessary to implement a U-turn, the behavioral complexity of this context is trivial, as shown in Figure 23.

The first state commanded the maneuver and monitored progress. Once the maneuver was completed, the state transitioned into the end state. In case of a collision threat, the stop state waited for the obstacles to clear. Under certain conditions, for example, when an obstacle was detected during the last back-up maneuver, it was possible to transition directly to the end state (i.e., the U-turn had completed sufficiently to resume operation).

3.3.4.5. *Zone*

The zone context was responsible for guiding the robot during entry into, exit out of, and driving while within zones. The controlling HSM is shown in

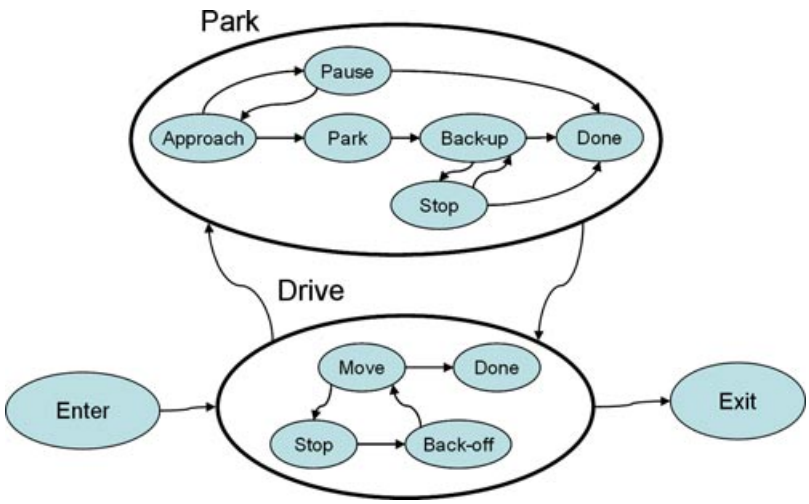

Figure 24. HSM for the zone context.

Figure 24, with hierarchical states shown inside each other. The entry state took over immediately upon reaching the way point that led into the zone. The drive state was designed to move the robot from the current location to any point in the zone, while avoiding other stationary and moving obstacles. When a parking task was necessary, the drive state moved the robot to a preparking spot, located adjacent to or on the extended centerline of a parking slot, then transitioned to the park state. When a parking task was not necessary, the drive state moved the robot to the zone exit and transitioned to the exit state.

While inside a zone the maximum speed was set to 5 mph (8 km/h), independent of the guidance provided in the MDF. All objects were set to avoid and all parking spots, except the target of the parking maneuver, were treated as obstacles, thus ensuring that the robot would not drive over them in accordance with California driving rules and DARPA instructions.

Even though the path planner could resolve the vast majority of situations it encountered, there are pathological cases during which the robot could "paint itself in a corner" (although this situation was never encountered at the NQE). The purpose of the back-off state was to perform a back-up maneuver that allowed the robot to get out of that situation. To compute the appropriate back-up maneuver, several geometrical approaches were tested in simulation. The most straightforward yet effective approach was to develop a set of deterministic back-up maneuvers and pick the one to use at random. If the new position did not allow progress, the system cycled through the stop and back-off states and a different maneuver was attempted. An illustration of the approach is shown in Figure 25. In this example, the maneuver is to back up $1^1/_2$ robot lengths and turn 30 deg to the left of the centerline.

When a parking maneuver was necessary, the drive state moved the robot to the prepark position and the system transitioned into the park state. A straightforward sequence of state changes within park guided the robot into and out of the parking spot.

This design was tested extensively on various parking lots on the UCF campus, and TeamUCF was pleasantly surprised at the robot's ability to navigate and park in very constrained parking lots that were filled with islands, obstacles, and parked vehicles, significantly more complex than even the challenging scenarios presented in the NQE.

3.3.4.6. Road

This context was responsible for guiding the robot from one way point on a lane to a subsequent way point on the same lane. Because the mechanics of generating and tracking the trajectory were handled elsewhere, this context was behaviorally simple. The upper level of the associated HSM is shown in Figure 26. Lacking any significant interaction with other objects, the drive state moves the robot along. If needed, road discovery was handled within the drive state.

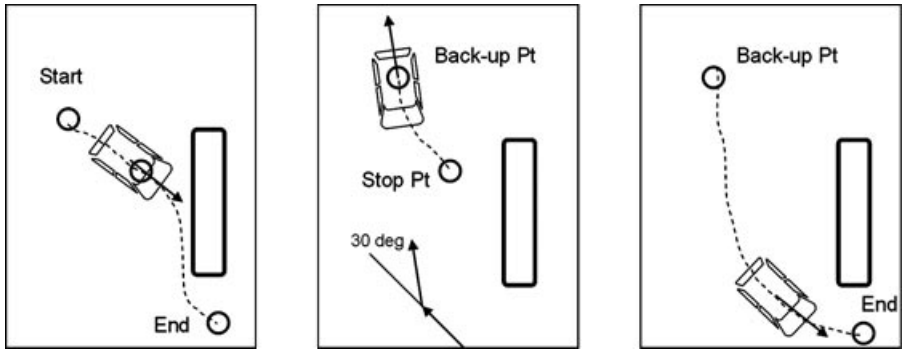

Figure 25. Example of back-off operation.

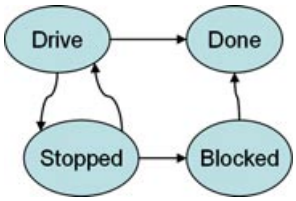

Figure 26. HSM for the road context.

Given an object, the robot must decide whether the object is something to follow or something to avoid. The approach utilized in Knight Rider was heavily biased by the characteristics of the data provided by the sensors. The approach utilized, illustrated in Figure 27, performed adequately given the competition rules. The top-level states represent object classification states and are not directly related to the behavioral states of the HSM. The states on the bottom reflect object disposition.

The initial condition is driving with no objects in sight and is shown by the left-most state. Once a new object appears it is classified as follow. An object that interrupts the baseline trajectory of the robot and is classified as follow will cause the robot to stop at a safe distance behind the lead object. If the robot stops for a certain period, the disposition of the lead object changes to avoid-abort, causing the path planner to plan around the object. As the robot goes around the object, one of two things can happen. The object can move, in which case its disposition reverts to follow, or the robot will travel past the object (Knight Rider used committed once the front of the robot reached the rear of the object), in which case the object's disposition is set to avoid-commit. Once an object becomes avoid-commit it cannot revert back to follow. Once behind the robot, its disposition is set to ignore, which eliminates it from consideration. After at least one object has been classified avoid-commit, any new object is automatically classified as avoid-commit (i.e., once the Knight Rider started passing, it continued to pass until it returned to the lane of travel). As objects were passed, they were labeled ignore, and once all objects were labeled ignore, the system reverted back to the start state with no objects. Figure 28 illustrates a series of four snapshots showing operation of this approach.

The first snapshot, located on the upper left, depicts the situation in which the object is set to follow, causing the robot to stop. After a brief pause, the object is set to avoid-abort, causing the passing maneuver. In the third snapshot, on the upper right, the object switches to avoid-commit. The final snapshot illustrates how a new object appearing at that point is automatically set to avoid-commit, providing a continuous passing maneuver.

3.3.4.7. Lane Change

The lane change context was responsible for guiding the robot while performing a lane change. Lane changes are planned during route generation and were defined with an approximate start and end location. Because of the a priori planning, lane changes were behaviorally rather simple, blending seamlessly between two drive contexts.

3.4. Path Planning

The path planner (PP) acted as the bridge between the tactical missions defined by AI and commands that

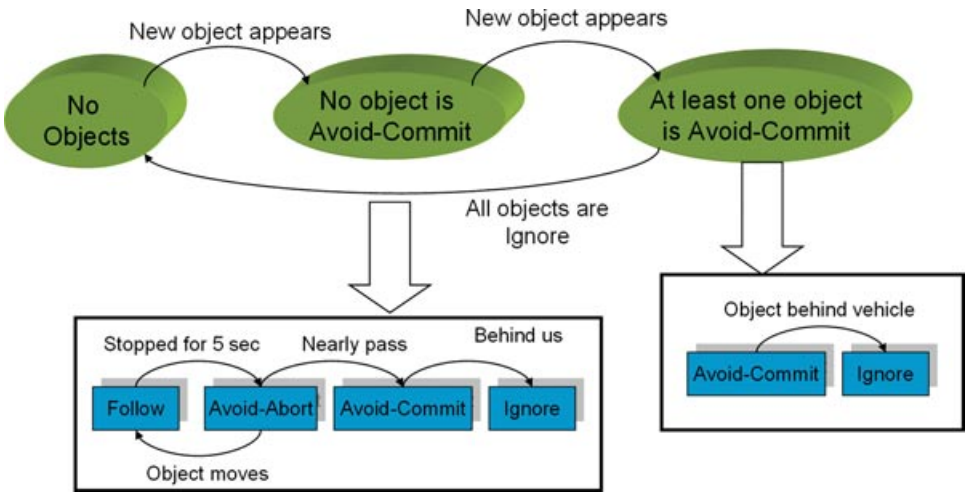

Figure 27. Handling object disposition.

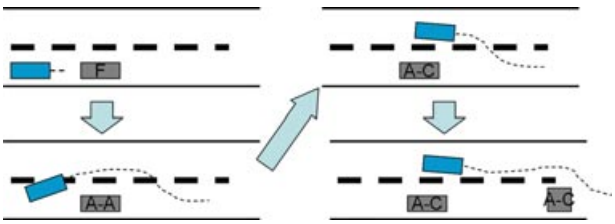

Figure 28. Example object disposition sequence.

could be executed by the autopilot. It performed this operation by effectively acting as a function call for the AI that would generate a dense list of way points from a sparse set of way points provided by AI. It ensured that the path generated by that dense list of points (0.5-m spacing) was kinematically feasible, met the explicit driving rules imposed by DARPA, and did not violate any constraints imposed by AI (such as speed limits or roadway boundaries). In addition to providing a dense path, the PP provided path length, estimated time to complete path, and avoidance information associated with every obstacle encountered on the path. By effectively acting as a function call, AI could explore different scenarios with the PP and select one to be forwarded to AI. Execution times were short enough that several scenarios could be explored in 100 ms.

The nominal problem for the PP was to create an in-bounds, kinematically feasible path from point P_0 to P_n, passing through intermediate points P_i (Figure 29). Feasibility includes speed and acceleration limits as well as boundary constraints. Initial work followed a unique analytical approach (Qu, Wang, & Plaisted, 2004; Yang, Daoui, Qu, Wang, & Hull, 2005) but was modified significantly as it was realized that assuming flexible objectives yielded substantially better performance in many scenarios. Specifically, the PP could explicitly violate objectives in the following manner:

- Any speed could be changed as long as the overall speed limits on segments were not violated and kinematic limits of the robot were not violated.
- Heading at a way point, if provided, was a suggestion and could be violated if required to keep a path in bounds. Direction of travel (forward or reverse) at a way point could not.
- Obstacles were classified as to be followed or to be avoided. If the desired path crossed a to-be-followed obstacle, the path was shortened based on the obstacle speed and type of area the obstacle was in (zone, road, near a stop sign, etc.). Basically, if a stopped obstacle was likely in a certain scenario, the robot could get closer to the obstacle than if it was unexpected.
- Intermediate way points could be moved perpendicular to the direction of travel if required to do so to avoid an obstacle.

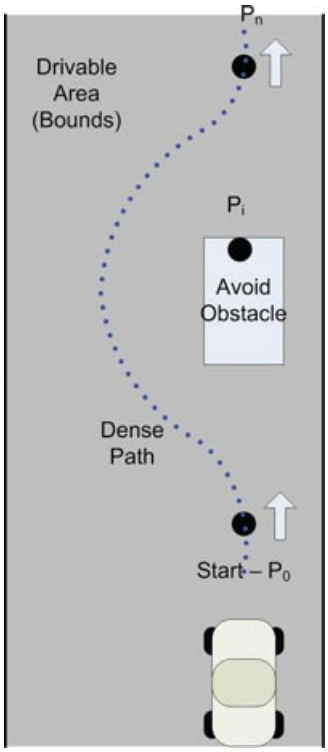

Figure 29. Dense path generation from sparse goals.

- The final way point could be moved along the direction of travel if required to do so because of an obstacle.
- Obstacles to be avoided are to be avoided by at least 1 m if possible, but if not possible a path as close as 0.25 m is acceptable.

The approach taken was one of iteratively generating piecewise, continuous first-derivative, cubic splines with updated control points as necessary to avoid obstacles and keep paths within bounds. Constraints were gradually relaxed if no solution was found. The use of splines generated smooth curves (although not necessarily optimum time of traversal paths), which could easily be evaluated for axial and lateral acceleration constraints.

PP operation when driving on a road or navigating a zone was fundamentally the same. Dense path information provided the path to follow until such time as goals, constraints, or obstacles changed. It also provided a convenient way for multiple systems to know where the robot was with respect to meeting its tactical objectives (namely, by associating the robot with the closest dense path point). Although the PP generated paths that could be driven by the vehicle, it was the autopilot's responsibility to follow the path generated by the PP.

3.5. Control Systems

The purpose of the low-level control system/autopilot (AP) was to physically actuate the plan put forth by the PP. The overall operational inputs to the system were a list of dense way points with 0.5-m spacing and the current navigation status. Each of these way points had an associated position, heading, and velocity, each of which should be physically realizable based on the robot dynamics. The overall requirements on the AP control systems were not near the vehicle or actuator limits, nor were their specific scoring parameters based on how precisely speed or steering was followed. Because of this, no optimal control system design was performed and only rudimentary modeling of actual subsystems (i.e., second-order response characteristics, rate and magnitude limits, etc.) was performed. Control system parameters were selected to mimic human drivers operating in similar circumstances. Whereas no formal comparison to multiple human drivers was performed, there is clearly significant diversity in driver performance. The team selected control system parameters based on one particular driver that we collectively judged to operate the vehicle in the most reasonable manner.

All control systems ran on a single-processor QNX using the real-time scheduling and interprocess communication systems of the operating system in order to minimize data latency and maximize predictability associated with operation.

3.5.1. Steering Control

Steering control (Figure 30) was provided by a follow-the-carrot controller (Barton, 2001) coupled with a state machine to handle three specific driving modes (normal, stopped, and three-point turn). In the latter case, tighter control of vehicle steering is essential to meet turn requirements. By passing a multipoint dense path between the PP and AP, the two systems did not need to maintain tight timing coupling and in fact the PP could operate significantly slower (e.g., 1 Hz demonstrated in testing) than the AP (20 Hz) during periods when the environment or obstacle mix was not rapidly changing.

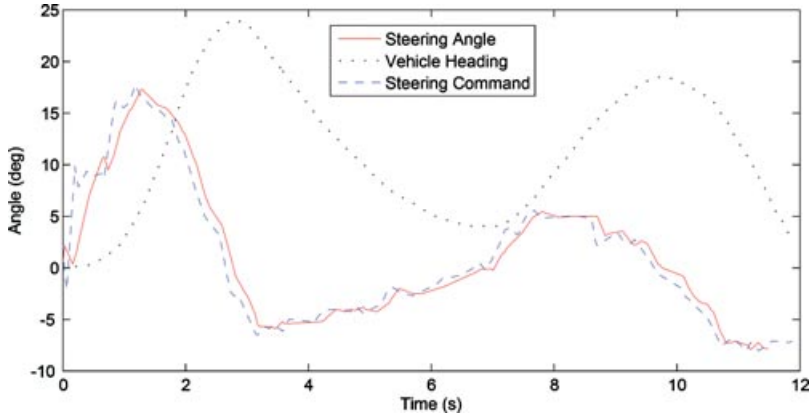

Figure 30. Typical steering controller performance.

At a 20-Hz update rate, the steering controller would 1) compute the closest dense path point to the current vehicle location, 2) look ahead on the path a fixed look-ahead "time" of 1.5 s, and 3) determine the effective carrot point. Using the carrot point, a heading was calculated between the current position of the robot and the position of the carrot point. This heading was then compared to the physical heading of the robot. The difference in these headings becomes the error for a proportional-integral (PI) controller, which feeds the steering wheel actuator. Steering command limits, steering rate limits, and integrator limits were included. In addition, integration was performed only during periods of the path when the path curvature was below a threshold. Special end-of-path logic (effectively linearly extending the path based on the last heading) avoided any steering discontinuities should path lengths become small or the vehicle be commanded to a stop. The objective here clearly is to keep the robot generally on the path without undue precision (10-cm error is tolerable).

The steering angle calculated through the follow-the-carrot method was passed to the Elmo motor controller. Through the actuation system, the desired steering angle is converted to an absolute encoder position, and the Elmo's internal PID controller physically maintained steering for any quick impulses or external stimulus feedback through the steering wheel from the environment. Stop-to-stop performance of the steering wheel took approximately 1.5 s.

Other path following schemes such as pure pursuit (Coulter, 1992) and the hybrid controller employed by Stanford in 2005 (Thrun et al., 2006) were explored, but the simplicity of follow the carrot coupled with its robustness and accuracy led to its selection.

3.5.2. Speed Control

Whereas the steering controller looked forward into the planned path to obtain a steering command, the speed controller used a linearly interpolated value of the current desired speed as an effective cruise control set point. Again, special end-of-path logic forced commanded speed to be zero at the end of a path and further forced a speed ramp down if the robot ever got so close to the end of a path it could not stop without violating acceleration constraints. Of course, PP logic should prevent this, but this strategy of hierarchical checking for basic system performance constraints proved to be critical during vehicle testing.

The subtle challenge associated with cruise control set points is determining when to brake and when to coast. TeamUCF utilized dual PI controllers, one for throttle and one for the brake, with a hysteresis crossover. The PI controller would have two internal states: throttle and brake. If the controller was in the throttle state, a positive value from the controller would be interpreted as a voltage to feed the throttle actuator. A negative value from the controller would represent cruising, which means that no throttle and no brake would be applied. A large enough negative value caused the system to transition to the brake state. In the brake state, a separate PI controller controlled the brake actuator. In this way the vehicle

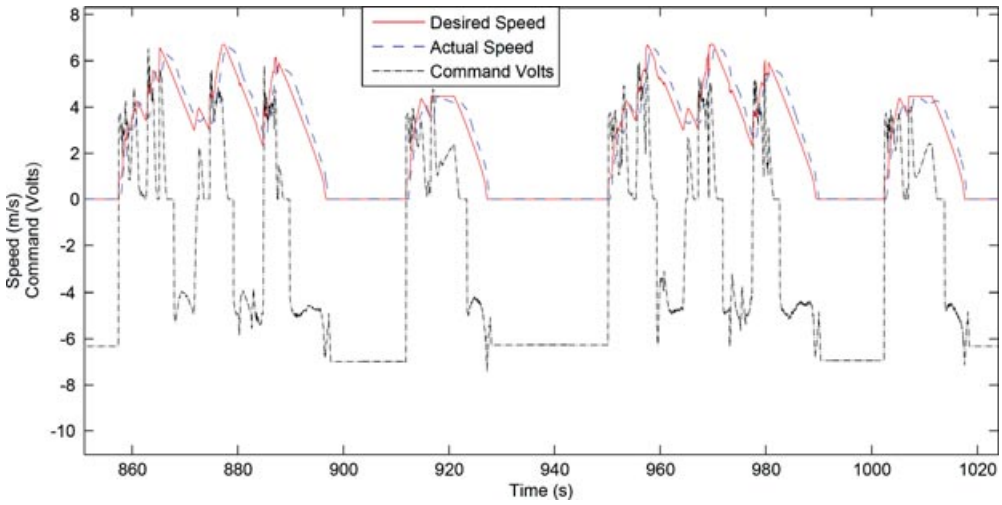

Figure 31. Speed repeatability over two laps of NQE test area A.

effectively operated in four states: 1) accelerating, 2) coasting throttle, 3) braking, and 4) coasting brake.

The performance and repeatability of the speed controller for subsequent runs over the same course can be seen in Figure 31. All elements of the system from the AI to the PP to the speed controllers were incredibly repeatable. This ability led to predictable behavior and simplified tasks for other systems. In the figure, the commanded volts curve illustrates throttle being applied (+ volts), brake applied (− volts), and coasting (0 volts). The performance here is typical, showing the vehicle operating within ∼60% of its capability (10-V peak).

3.5.3. Drive State Control

Because the underlying robot vehicle was an automatic, no shifting was required; however, the system still needed to be brought into the proper gear (park, forward, or reverse). The main function of the shift controller was to ensure that there was enough time between shifting and further timing operation for the robot to be safe. For example, if the robot was currently moving 1 m/s and in the forward gear and was just commanded a −1 m/s speed, the state would switch from forward to prereverse. Prereverse would smoothly stop the robot and wait until the speed was 0 m/s for a second. The brake would then be fully applied, and the state would switch to shift-reverse. In shift-reverse, the controller would send the voltage to physically actuate the robot to reverse and wait for another second to ensure that the shift finished. At the end of that time, the system would then switch to the reverse state and reset the dual PI integrator error.

3.5.4. E-Stop Pause

The pause system also consisted of a series of state changes. The digital output line of the DARPA E-stop device was read on an I/O pin and debounced to ensure that a false reading was unlikely. Upon receiving a pause command, the vehicle would be brought to a stop and the appropriate combination of sirens and lights activated. Further, once in pause, a run command would cause the robot to wait 5 s before actually beginning operation.

3.6. ICE

The main communication system used throughout the robot was provided by the ICE by ZeroC. This highly efficient middleware package allowed the robot's software to be distributed over a heterogeneous network of machines. Data types were handled though ICE's mechanisms that allowed data to be shared to multiple destinations regardless of operating system or programming language. TeamUCF utilized the publish-and-subscribe model in which multiple programs would be able either to request the most recent information by name or execute

when new information was available. This framework worked well for the distributed architecture used, and ICE overhead was never a factor even when publishing large sensor data structures.

Perhaps most important, the module independence provided by ICE provided the ability to test the robot offline through simulation. Simulated modules that would take the same robot actuation signals over ICE and implement them on the actual robot were gathered and run through a dynamic physics simulation. Intelligence, planning, control system algorithms, and sensor processing algorithms could receive simulated or real data while still providing outputs in real time. In this way there was only one version of production and test code.

4. PROJECT PROCESS

TeamUCF participated in the original proposal submission but failed to gain track A status. Undeterred, the team executed on a capabilities-driven implementation approach. At each stage of this approach, the key capabilities to be demonstrated next were determined. These capabilities were occasionally defined by DARPA (as in the case of a site visit) but were more likely to be defined by TeamUCF leadership. All systems were developed in parallel and to the level necessary to demonstrate the capability. In this way, incremental testing of the robot was performed for many months prior to the NQE. The downside of the approach is, of course, that full operational capability was invariably ready only just prior to the NQE.

4.1. Simulation and Modeling

Simulation was critical to the overall success of the project. The ability to test the robot in a multitude of scenarios, virtually, allowed different team members to test changes quickly on their own computers or over a network of computers. Furthermore, with a small team effectively responsible for both software development and robot testing and a limited window for testing, there were simply not enough hours in the day to conduct all the desired tests without simulation.

Coupling the simulation environment with a source management tool, in this case subversion (SVN), allowed problems that were detected on the real robot to be checked in as data files, sent to distributed team members, replicated in simulation, and resolved. That error could then be corrected in simulation, the source code checked in, and operation corrected on the robot, typically the same day. TeamUCF maintained no software laboratory or significant facility for any development activity.

The key to this ability was a product of the ICE middleware distribution and the modular nature of the software design. The ICE architecture allowed the real robot code to be used with virtual sensors. This software-in-the-loop scheme of sensor replacement was implemented by creating a package of threads, each tasked with publishing realistic data. Each of the threads then published to the middleware level for use by the other modules.

The synthetic sensor fusion module had the ability to inject moving obstacles with complex obstacle behaviors into the virtual world in the same fashion as the postprocessed data that the actual sensor fusion would develop from the laser scans and Doppler data. All synthetic data generation had the ability to be perfect ground truth, to contain random fluctuations consistent with the noise levels measured in sensor systems, or, perhaps most important, to play back an actual vehicle log of the same data.

The synthetic E-stop module reflected the DARPA E-stop unit that could send the different pause and disable commands. On the basis of experiences in 2005, TeamUCF felt it was essential to test the operational effects of these commands on the rest of the system. The pauses were known to be a factor in the Urban Challenge due to the number of robots on the course, and TeamUCF spent significant time pausing and restarting the robot in as many different scenarios as possible.

Synthetic kinematics was generated from a robot dynamics model using Ackermann steering with second-order response functions with rate and position limiters for all actuators. Synthetic navigation was generated from this ground truth by adding appropriate filtered white noise that closely matched the performance of the actual sensor systems. Most important, these navigation and control processes contained appropriate process delays, modeled via FIFOs, to account for transport-and-process delay observed in the system. An early software error caused an asynchronous clock skew that was debugged using this technique and resolved with a combination of error correction and more fault-tolerant algorithms.

All of the data produced by these systems was published to ICE. A graphical visualizer, Vevis, was developed in OpenGL to display the robot and environment in real time. The software developer could

then view the entire process unfold and observe the actions of the robot from movement to turn signals to direction of the radar. This overview could then display the route planned by the AI, the dense points created by the PP, and the movements produced by the AP. Vevis allowed the team to zoom in on specific regions, load RNDF and MDF for simulations of entire runs, or load the maps files on available textures to verify operation of the calculated road network. Because Vevis requested all usable information from ICE, this system was also used in real time on the actual robot.

4.2. Testing Methodology

Although simulation was key to TeamUCF's success, the team spent an equal amount of time on the actual robot. TeamUCF's test site was unfortunately available only after hours, and therefore the majority of tests were conducted in the evening, which of course has added benefits when testing in the summer in Florida.

The test philosophy dictated that the team would run numerous shorter duration tests that could be quickly validated via simulation. A typical testing evening consisted of 6 h of tests, consistent with the amount of time expected at the final event, but the vehicle was never tested in a single 6-h mission. Whereas the majority of the test runs were performed at night (for safety and facility access reasons), extensive test runs also occurred during the day. This included the DARPA site visit. From daytime test runs and through our experience from the DARPA Grand Challenge 2005, we concluded that laser performance (from dazzle) was impacted but did not significantly degrade the obstacle detection capabilities of the algorithms employed.

5. NQE AND RACE RESULTS

The NQE and final event were held at George Air Force Base in Victorville, California. DARPA spent considerable time and energy preparing the facility to act as a safe but challenging test environment for the robots. Thirty-five semifinalists were invited to participate in the NQE, including teams from all over the United States and several teams with a large international contingency. TeamUCF had arguably spent the smallest amount of money and had the smallest team to make it to the finals and was possibly the smallest team in the semifinals as well.

5.1. NQE

The details of the testing to be performed at the NQE were unknown to the participants until they arrived at the event. Shortly after arriving, teams learned that the qualifying event would consist of a series of missions in three test areas, creatively named A, B and C, which stressed different aspects of the robot. Each team would have two chances to perform each test. Test area A was visible to all team members, but the details of test areas B and C were not. Teams were not permitted to drive on any of the test areas or for that matter much of the Air Force Base. TeamUCF was assigned the test areas in order B, C, A.

Figure 32 illustrates the layout of test area B and also illustrates the simulation and visualization software utilized by TeamUCF. This is a screen display from either inside the vehicle or the simulation: they are identical. The baseball diamond near the center of the figure gives some sense of scale. A series of k-rail launch chutes at the upper left of the figure defined the launch location. The vehicle immediately enters a zone driving area with no specific way point information other than an exit goal at the bottom left of the zone. Upon leaving the zone the vehicle must traverse a narrow pathway bounded by k-rails and negotiate a roundabout, eventually out into a double-cloverleaf road network. At the center of each cloverleaf is another zone area, with the bottom zone modeling a parking lot and requiring a parking maneuver. In the center of each zone, and not shown, are two large circular barricades that were to be detected and avoided. There were numerous other static obstacles on the course to be avoided. There were no moving obstacles. The mission wound through much of the course with the robot required to return back to the starting location, completing a course of about 6.5 km in 30 min.

The challenges presented by test area B were effectively as follows:

- Navigate a zone with no way point information
- Navigate over a relatively long distance and relatively long time
- Navigate in the presence of sparse way points (note upper right portion of the figure)
- Navigate through a complex field of static obstacles
- Navigate narrow roads with barriers on either side of the road

Figure 32. Test area B: Driving, parking, and obstacle avoidance.

- Navigate stop signs
- Park with nearby parking spots occupied by vehicles

TeamUCF's first attempt at test area B resulted in the robot making it through the majority of the course (∼5 km) in approximately 20 min after successfully navigating the parking maneuver and a "gauntlet" of parked cars. At about this time, the sensor fusion algorithm failed due to a software bug in a polygon clipping module. The vehicle effectively lost all forward-looking sensors and crashed into one of the barricades, resulting in minor damage to a cable. TeamUCF was allowed to restart the vehicle from here, but we did not complete the course. Our second attempt at test area B was completed successfully in just under 19 min, one of the fastest qualifying times.

Test area B was an amazing awakening event for TeamUCF as we watched the robot leave our sight to enter into the heart of area B. We realized only then that this was the first time, in all of our testing, we had ever let the robot out of our sight.

Test area C (Figure 33) was designed to test stop sign and rerouting behavior. The MDF for the robot required a path be taken around the outer loop of the "belt buckle." Each time the robot reached a crossing four-way stop, a different configuration of cars was presented. The robot's objective was to correctly determine precedence and continue the mission in the correct order. After completing a series of loops, the course was adjusted, and a blockage inserted on the bottom loop of the buckle. The robot needed to reroute and determine a path to a checkpoint on the other side of the blockage. This proved problematic for many robots, including initially TeamUCF. A strict interpretation of rules would imply that such a point is unreachable in normal driving because it would require a U-turn on the far side of the barricade, but a U-turn is legal only on a blocked road, so the robot would have to assume that the blockage remained in place. DARPA rules (and in fact actions on this course) indicated that blockages could not be assumed to be static. Upon understanding these new constraints, TeamUCF was able to redefine the routing behavior to make this checkpoint.

Figure 33. Test area C: Stop sign and rerouting.

One of the more interesting things in this figure is the misregistration between the ground truth way point data and the imagery data provided by DARPA to all participants. Although useful for determining conceptually what a course looked like, the imagery data were of poor enough quality to not be usable for any type of premission planning or environmental modeling. Note that data from several popular Web-based mapping tools are similarly inaccurate.

Test area A (Figure 34) proved to be the most interesting area, perhaps because it was fully visible to spectators and perhaps because it was specifically designed to promote robot–manned vehicle interaction. The fundamental objective was to have the robot complete as many loops of the right-hand side of the course as possible in 30 min. Traffic crossed in front of the robot at the top of the course, and the vehicle was required to merge into traffic from the stop sign at the bottom center of the course. Traffic speed was relatively modest at ~10 mph (16 km/h), but traffic configurations were continually altered by the drivers, and unless the robot was relatively aggressive none of the maneuvers could be performed without some close calls with the manned vehicles (Figure 35). TeamUCF completed 17 transits of the course in under 30 min and terminated the run early in order to avoid pushing "our luck." This was among the most laps performed.

5.2. Finals

By successfully completing all three test areas, TeamUCF earned a place in the finals of the Urban Challenge along with 10 other competitors. DARPA narrowed the field of finalists from the initially stated goal of 20 to only 11 competitors.

Figure 34. Test area A: Merging and crossing traffic.

The final event took place on November 3, 2007, and was composed of a series of three missions covering a distance of 60 miles (97 km) through a complex urban environment and driving time limited to a total of 6 h. Test tracks A and B of the NQE were incorporated as subsets into the final road network, but the network extended to other new areas with significantly more elevation change than the relatively flat NQE test areas. The three missions were designed to demonstrate all of the scenarios previously tested, although in a somewhat less stressing manner, plus some novelties. In addition to some 50 human-driven traffic vehicles, all finalist robots were on the track at the same time, creating for TeamUCF the never-tested scenario of encountering live robot traffic. To add a level of complexity, DARPA announced a day before the final event that one section of the urban course would be a steep unpaved road negotiating an elevation difference of 50 m. TeamUCF was surprised to find off-road performance tested in a contest labeled as "Urban."

During the first 30 min of the race, Knight Rider behaved as expected, mastering encounters with other robots without any problems (Figure 36) and driving road segments reliably and repeatably. At 9:11 a.m. the robot got stuck at a stop sign, not entering the intersection even when all other pockets were empty. Most likely a misreading by the sensors produced a phantom obstacle in the intersection, essentially deadlocking Knight Rider. The situation resolved itself after a few minutes when another vehicle entered the pocket and cleared the obstacle. At 9:42 a.m. "Little Ben" from the University of Pennsylvania came within a few centimeters of Knight Rider when switching lanes after passing TeamUCF's chase vehicle.

After 19.8 km (~12.3 miles) and a running time of 2 h, 7 min, and 20 s, the Knight Rider GPS/INS returned a NaN (not a number) for the latitude and longitude of the robot. The manufacturer-provided communication library used to read the UDP packages from the GPS reported a nonsuspecting "data valid"

Figure 35. Illustration of crossing traffic.

Figure 36. Knight Rider encountering MIT in the traffic circle.

for these values. In the IEEE floating-point standard, any arithmetic operation involving NaN always results in NaN, and any numerical comparison with it fails. The invalid data cascaded through the system but had the most detrimental impact on the steering controller. In the steering controller, the robot's position is used to calculate the angle to the carrot point as a set point for the PI controller. The integral part of this controller preserved the value for subsequent iterations, essentially locking the value into the system. Confronted with this, the digital servo drive locked the steering wheel in the center position, leading the robot to deviate from the road, jumping a curb and driving toward an abandoned house, eventually stopping in front of a wall, where it stayed paused for the next 6 h (Figure 37) before the team was allowed to recover it. TeamUCF was officially retired from the DARPA Urban Challenge at 10:38 a.m.

6. DISCUSSION

The experience gained during the Urban Challenge competition has been invaluable to advancing TeamUCF's capabilities with autonomous robotic vehicles operating in an urban environment. The performance of a number of teams clearly indicates that commercial autonomous vehicle operation is closer to reality than many expect.

Whereas TeamUCF encountered a series of mechanical issues during the NQE, none of these issues

Figure 37. Knight Rider paused at his final resting place right in front of a house.

severally hampered performance. The overall competition was not particularly stressful on the robots from a mechanical point of view. Furthermore, all sensors operated within expected performance bounds throughout the competition. This was partially because adverse environmental conditions that could have impacted sensor performance were reduced by the choice of venue and the time of year of testing. The use of relatively simple algorithms, robust simulation tools, and partitioning of the control system in the manner used contributed to the ability to make the few minor modifications that were necessary during the NQE. In hindsight, the choice of the platform for the robot, the choice of sensor systems, and the overall control approach were good ones and ones the team would use again.

In many ways the final event was easier than the tests required to be passed to qualify for the finals. The final event focused on a robot's ability to repeatedly perform a series of moderately challenging missions over the course of hours. Teams that had significant experience with long-duration tests fared better than those that did not. The criticality of long-duration testing cannot be underestimated, but the implications for preevent test area configuration should be understood. The availability of a test area where a vehicle can be driven more than 10 km and for hours without fear of unintended interaction with other vehicles was a deciding factor in determining the outcome of this event. A small test area offered the ability to investigate "scenarios" or demonstrate the robot's ability to meet virtually every individual objective. Testing at a nearby parking lot during evenings while at the NQE allowed specific algorithms associated with test area A to be validated and refined. Testing at an off-road site verified the off-road capability of the robot prior to participation in the final event. Several teams took advantage of this scenario testing. Nevertheless, scenario testing was woefully inadequate in verifying the robot's long-term performance. In hindsight, TeamUCF might have fared better in the final event by verifying that the robot could drive around a circular course for 8 h straight rather than verifying off-road performance.

The criticality of GPS or GPS/INS systems cannot be underestimated. On the day prior to the final event, finalists were asked to launch their robots from the starting chutes and make a relatively simple loop course ostensibly to verify starting procedures and timing. A robot that had performed virtually flawlessly in previous tests nearly collided with two other

robots that were stopped behind the start line. The team with the malfunctioning robot claimed that the issue was traceable to a malfunctioning GPS. On the day of the final event, the pole-setting team failed to launch on time, again due to a stated GPS malfunction attributed to interference from a large TV screen near the vehicle. TeamUCF's failure to complete the final event is directly attributable to a GPS failure. A significant GPS outage for any team would likely have crippled that team.

Several other items are noteworthy:

- SICK laser scanners are highly reliable and robust measurement devices, but with relatively limited operational range. Certain lighting conditions or obstacle types bring those ranges well below 50 m, making these sensors questionable for vehicle speeds much above those demonstrated in this challenge.
- The Oxford RT3000 GPS/INS is highly capable and accurate; however, external data validity checking, separate from the software tools provided by the vendor, must be performed to ensure that the rare data error does not have catastrophic consequences.
- TeamUCF struggled with camera-based vision systems primarily because of lighting conditions and obstacle diversity, but other systems made up for these deficiencies.
- ICE worked well for interprocess communications and effectively supported hardware-in-the-loop capability. Coupled with SVN, these open-source tools provide an incredible software development environment for robotic systems.
- A large-scale test site with essentially unlimited access is essential for adequate testing. Long-duration performance can be adequately tested only at such a facility. TeamUCF conducted the majority of its testing on a large open parking deck, at night.
- TeamUCF and many other teams pursued this effort as a competition, with the goal of meeting the specific objectives of the competition. As such, many systems and approaches were tailored to specifically follow the rules as defined by DARPA.
- Money matters, but only if you use it to allow robust vehicle testing.
- NaN fails every test.

ACKNOWLEDGMENTS

TeamUCF thanks the University of Central Florida College of Engineering and Computer Science and Coleman Technologies, Inc., for generous sponsorship and Old Dominion University for flexibility in providing the time for Dr. Papelis to complete a project he started while at UCF. Furthermore, TeamUCF thanks Richard "Ed" Johnson from the University of Central Florida for his contributions in designing the 3-D laser scanner assemblies.

REFERENCES

Adams, M. (2001). On-line gradient based surface discontinuity detection for outdoor scanning range sensors. Proceedings. 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2001, Maui, HI (Volume 3, pp. 1726–1731).

Barton, M. (2001). Controller development and implementation for path planning and following in an autonomous urban vehicle. Undergraduate thesis, University of Sydney, Sydney, Australia.

Boer, E., & Hoedemaeker, M. (1998). Modeling driver behavior with different degrees of automation: A hierarchical decision framework for interactive mental models. In Proceedings of the 17th European Annual Conference on Human Decision Making and Manual Control, Valenciennes, France.

Coulter, R. C. (1992). Implementation of the pure pursuit path tracking algorithm (Tech. Rep. CMU-RI-TR-92-01). Pittsburgh, PA: Robotics Institute, Carnegie Mellon University.

Cremer, J., Kearney, J., & Papelis, Y. (1995). HCSM: A framework for behavior and scenario control in virtual environments. ACM Transactions on Modeling and Computer Simulation, 5(3), 242–267.

Harper, D., Hua, D. K., Foroosh, D. H., Leonessa, D. A., Qu, D. Z., Pillat, R., Norvell, D., Santiago, S., Collins, T., Stein, G., Stickler, S., Decker, G., Andres, R., Shen, Y., Chen, H., & Xie, F. (2005). DARPA Grand Challenge 2005 (Tech. Rep.). Orlando: University of Central Florida.

Michon, J. (1985). A critical view of driver behaviour models: What do we know, what should we do? In L. Evans & R. Schwing (Eds.), Human behaviour and traffic safety. New York: Plenum.

Moravec, H. (1988). Sensor fusion in certainty grids for mobile robots. AI Magazine, 9(2), 61–74.

Papelis, Y., & Ahmad, O. (2001). A comprehensive microscopic autonomous driver model for use in high-fidelity driving simulation environments. In Proceedings of 81st Annual Meeting of the Transportation Research Board, Washington, DC.

Qu, Z., Wang, J., & Plaisted, C. E. (2004). A new analytical solution to mobile robot trajectory generation in the presence of moving obstacles. IEEE Transactions on Robotics, 20, 978–993.

Surmann, H., Nuechter, A., & Hertzberg, J. (2003). An autonomous mobile robot with a 3D laser range finder for 3D exploration and digitalization of indoor environments. Robotics and Autonomous Systems, 45, 181–198.

Thrun, S., Burgard, W., & Fox, D. (2005). Probabilistic robotics. Intelligent robotics and autonomous agents. Cambridge, MA: MIT Press.

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., Stand, 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., & Mahoney, P. (2006). Stanley: The robot that won the DARPA Grand Challenge. Journal of Field Robotics: Special Issue on the DARPA Grand Challenge, Part 2, 23(9), 661–692.

Wulf, O., & Wagner, B. (2003). Fast 3D-scanning methods for laser measurement systems. International Conference on Control Systems and Computer Science (CSCS14), July 2–5, 2003, Bucharest, Romania.

Yang, J., Daoui, A., Qu, Z., Wang, J., & Hull, R. (2005). An optimal and real-time solution to parameterized mobile robot trajectories in the presence of moving obstacles. In 2005 IEEE International Conference on Robotics and Automation, Barcelona, Spain (pp. 4423–4428).