Junior: The Stanford Entry in the Urban Challenge

Michael Montemerlo[1], Jan Becker[4], Suhrid Bhat[2], Hendrik Dahlkamp[1], Dmitri Dolgov[1], Scott Ettinger[3], Dirk Haehnel[1], Tim Hilden[2], Gabe Hoffmann[1], Burkhard Huhnke[2], Doug Johnston[1], Stefan Klumpp[2], Dirk Langer[2], Anthony Levandowski[1], Jesse Levinson[1], Julien Marcil[2], David Orenstein[1], Johannes Paefgen[1], Isaac Penny[1], Anna Petrovskaya[1], Mike Pflueger[2], Ganymed Stanek[2], David Stavens[1], Antone Vogt[1], and Sebastian Thrun[1]

[1]Stanford Artificial Intelligence Lab, Stanford University, Stanford CS 94305
[2]Electronics Research Lab, Volkswagen of America, 4009 Miranda Av., Palo Alto, CA 94304
[3]Intel Research, 2200 Mission College Blvd., Santa Clara, CA 95052
[4]Robert Bosch LLC, Research and Technology Center, 4009 Miranda Ave, Palo Alto, CA 94304

Abstract

This article presents the architecture of Junior, a robotic vehicle capable of navigating urban environments autonomously. In doing so, the vehicle is able to select its own routes, perceive and interact with other traffic, and execute various urban driving skills including lane changes, U-turns, parking, and merging into moving traffic. The vehicle successfully finished and won second place in the DARPA Urban Challenge, a robot competition organized by the U.S. Government.

1 Introduction

The vision of self-driving cars promises to bring fundamental change to one of the most essential aspects of our daily lives. In the U.S. alone, traffic accidents cause the loss of over 40,000 people annually, and a substantial fraction of the world's energy is used for personal car-based transportation (U.S. Department of Transportation, 2005). A safe, self-driving car would fundamentally improve the safety and comfort of the driving population, while reducing the environmental impact of the automobile.

In 2003, the Defense Advanced Research Projects Agency (DARPA) initiated a series of competitions aimed at the rapid technological advancement of autonomous vehicle control. The first such event, the "DARPA Grand Challenge," led to the development of vehicles that could confidently follow a desert trail at average velocities nearing 20mph (Buehler et al., 2006). In October 2005, Stanford's robot "Stanley" won this challenge and became the first robot to finish the 131-mile long course (Montemerlo et al., 2006). The "DARPA Urban Challenge," which took place on November 3, 2007, brought about vehicles that could navigate in traffic in a mock urban environment.

The rules of the DARPA Urban Challenge were complex (DARPA, 2007). Vehicles were provided with a digital street map of the environment, in the form of a *Road Network Description File*, or

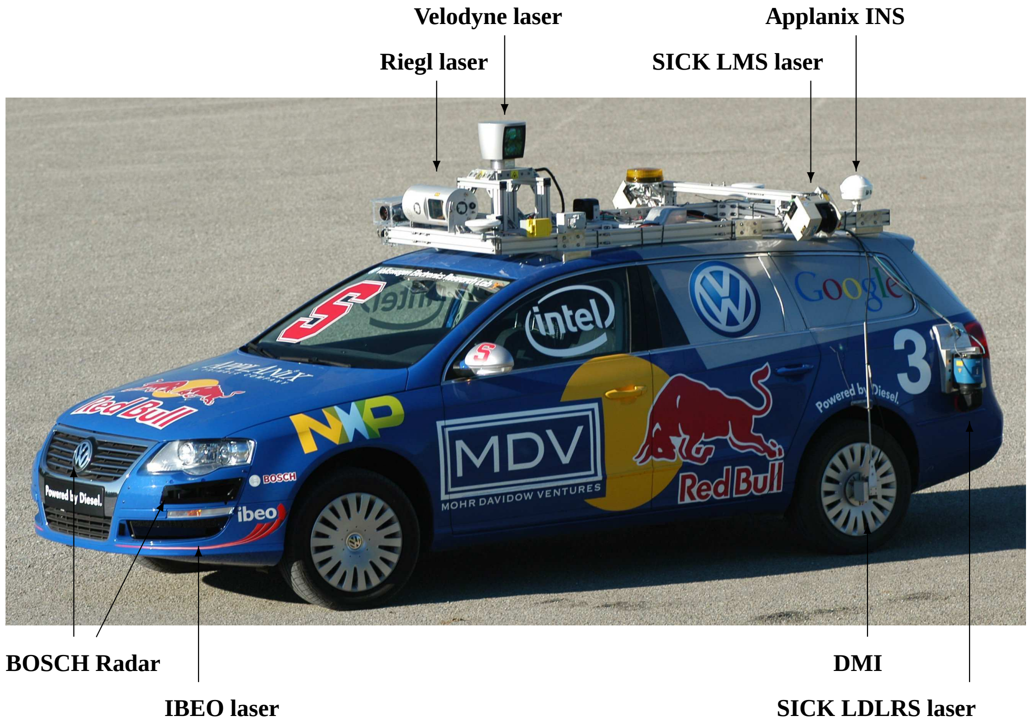


Figure 1: Junior, our entry in the DARPA Urban Challenge. Junior is equipped with five different laser measurement systems, a multi-radar assembly, and a multi-signal inertial navigation system, as shown in this figure.

RNDF. The RNDF contained geometric information on lanes, lane markings, stop signs, parking lots, and special checkpoints. Teams were also provided with a high-resolution aerial image of the area, enabling them to manually enhance the RNDF before the event. During the Urban Challenge event, vehicles were given multiple missions, defined as sequences of checkpoints. Multiple robotic vehicles carried out missions in the same environment at the same time, possibly with different speed limits. When encountering another vehicle, each robot had to obey traffic rules. Maneuvers that were specifically required for the Urban Challenge included: passing parked or slow-moving vehicles, precedence handling at intersections with multiple stop signs, merging into fast-moving traffic, left turns across oncoming traffic, parking in a parking lot, and the execution of U-turns in situations where a road is completely blocked. Vehicle speeds were generally limited to 30mph, with lower speed limits in many places. DARPA admitted eleven vehicles to the final event, of which the present vehicle was one.

“Junior,” the robot shown in Figure 1, is a modified 2006 Volkswagen Passat Wagon, equipped with five laser rangefinders (manufactured by IBEO, Riegl, Sick, and Velodyne), an Applanix GPS-aided inertial navigation system, five BOSCH radars, two Intel quad core computer systems, and a custom drive-by-wire interface developed by Volkswagen’s Electronic Research Lab. The vehicle has an obstacle detection range of up to 120 meters, and reaches a maximum velocity of 30mph, the maximum speed limit according to the Urban Challenge rules. Junior made its driving decisions through a distributed software pipeline that integrates perception, planning, and control. This software is the focus of the present article.

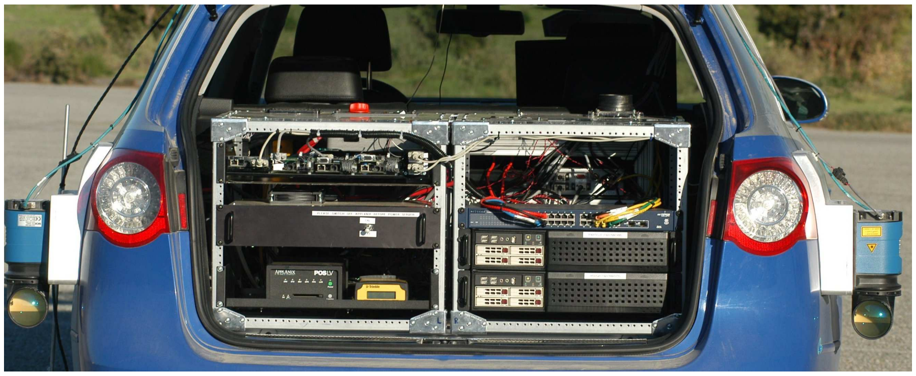


Figure 2: All computing and power equipment is placed in the trunk of the vehicle. Two Intel quad core computers (bottom right) run the bulk of all vehicle software. Other modules in the trunk rack include a power server for selectively powering individual vehicle components, and various modules concerned with drive-by-wire and GPS navigation. A 6 DOF inertial measurement unit is also mounted in the trunk of the vehicle, near the rear axle.

Junior was developed by a team of researchers from Stanford University, Volkswagen, and its affiliated corporate sponsors: Applanix, Google, Intel, Mohr Davidow Ventures, NXP, and Red Bull. This team was mostly comprised of the original Stanford Racing Team, which developed the winning entry "Stanley" in the 2005 DARPA Grand Challenge (Montemerlo et al., 2006). In the Urban Challenge, Junior placed second, behind a vehicle from Carnegie Mellon University, and ahead of the third-place winner from Virginia Tech.

2 Vehicle

Junior is a modified 2006 Passat wagon, equipped with a 4-cylinder turbo diesel injection engine. The 140 hp vehicle is equipped with a limited-torque steering motor, an electronic brake booster, electronic throttle, gear shifter, parking brake, and turn signals. A custom interface board provides computer control over each of these vehicle elements. The engine provides electric power to Junior's computing system through a high-current prototype alternator, supported by a battery-backed electronically controlled power system. For development purposes, the cabin is equipped with switches that enable a human driver to engage various electronic interface components at will. For example, a human developer may choose the computer to control the steering wheel and turn signals, while retaining manual control over the throttle and the vehicle brakes. These controls were primarily for testing purposes; during the actual competition, no humans were allowed inside the vehicles.

For inertial navigation, an Applanix POS LV 420 system provides real-time integration of multiple dual-frequency GPS receivers which includes a GPS Azimuth Heading measurement subsystem, a high-performance inertial measurement unit, wheel odometry via a distance measurement unit (DMI), and the Omnistar satellite-based Virtual Base Station service. The real-time position and orientation errors of this system were typically below 100 cm and 0.1 degrees, respectively.

Two side-facing SICK LMS 291-S14 sensors and a forward-pointed RIEGL LMS-Q120 laser sensor provide measurements of the adjacent 3-D road structure and infrared reflectivity measurements of the road surface for lane marking detection and precision vehicle localization.

For obstacle and moving vehicle detection, a Velodyne HDL-64E is mounted on the roof of the vehicle. The Velodyne, which incorporates 64 laser diodes and spins at up to 15 Hz, generates dense range data covering a 360 horizontal field-of-view and a 30 degree vertical field-of-view. The Velodyne is supplemented by two SICK LDLRS sensors mounted at the rear of the vehicle, and two IBEO ALASCA XT lidars mounted in the front bumper. Five BOSCH Long Range Radars (LRR2) mounted around the front grill provide additional information about moving vehicles.

Junior's computer system consists of two Intel quad core servers. Both computers run Linux, and they communicate over a gigabit ethernet link.

3 Software Architecture

Junior's software architecture is designed as a data driven pipeline in which individual modules process information asynchronously. This same software architecture was employed successfully by Junior's predecessor Stanley in the 2005 challenge (Montemerlo et al., 2006). Each module communicates with other modules via an anonymous publish/subscribe message passing protocol, based on the Inter Process Communication Toolkit (IPC) (Simmons and Apfelbaum, 1998).

Modules subscribe to message streams from other modules, which are then sent asynchronously. The result of the computation of a module may then be published to other modules. In this way, each module is processing data at all times, acting as a pipeline. The time delay between entry of sensor data into the pipeline to the effect on the vehicle's actuators is approximately 300ms. The software is roughly organized into five groups of modules.

- **sensor interfaces –** The sensor interfaces manage communication with the vehicle and individual sensors, and make resulting sensor data available to the rest of the software modules.
- **perception modules –** The perception modules segment the environment data into moving vehicles and static obstacles. They also provide precision localization of the vehicle relative to the digital map of the environment.
- **navigation modules –** The navigation modules determine the behavior of the vehicle. The navigation group consists of a number of motion planners, plus a hierarchical finite state machine for invoking different robot behaviors and preventing deadlocks.
- **drive-by-wire interface –** Controls are passed back to the vehicle through the drive-by-wire interface. This module enables software control of the throttle, brake, steering, gear shifting, turn signals, and emergency brake.
- **global services –** A number of system level modules provide logging, time stamping, message passing support, and watchdog functions to keep the software running reliably.

Table 1 lists the actual processes running on the robot's computers during the race event, and Figure 3 shows a overview of the data flow between modules.

Process name	Computer	Description
PROCESS-CONTROL	1	starts and restarts processes, adds process control via IPC
APPLANIX	1	Applanix interface (via IPC).
LDLRS1 & LDLRS2	1	Sick LDLRS laser interface (via IPC).
IBEO	1	IBEO laser interface (via IPC).
SICK1 & SICK2	1	Sick LMS laser interfaces (via IPC).
RIEGL	1	Riegl laser interface (via IPC).
VELODYNE	1	Velodyne laser interface (via IPC and shared memory). This module also projects the 3d points using Applanix pose information.
CAN	1	CAN bus interface
RADAR1 - RADAR5	1	Radar interfaces (via IPC).
PERCEPTION	1	IPC/Shared Memory interface of Velodyne data, obstacle detection, dynamic tracking and scan differencing
RNDF_LOCALIZE	1	1D localization using RNDF
HEALTHMON	1	logs computer health information (temperature, processes, CPU and memory usage)
PROCESS-CONTROL	2	start/restarts processes and adds process control over IPC
CENTRAL	2	IPC-server
PARAM_SERVER	2	central server for all parameters
ESTOP	2	IPC/serial interface to DARPA E-stop
HEALTHMON	2	monitors the health of all modules
POWER	2	IPC/serial interface to power-server (relay card)
PASSAT	2	IPC/serial interface to vehicle interface board
CONTROLLER	2	vehicle motion controller
PLANNER	2	path planner and hybrid A* planner

Table 1: Table of processes running during the Urban Challenge.

4 Environment Perception

Junior's perceptual routines address a wide variety of obstacle detection and tracking problems. Figure 4a shows a scan from the primary obstacle detection sensor, the Velodyne. Scans from the IBEO lasers, shown in Figure 4b, and LDLRS lasers are used to supplement the Velodyne data in blind spots. A radar system complements the laser system as an early warning system for moving objects in intersections.

4.1 Laser Obstacle Detection

In urban environments, the vehicle encounters a wide variety of static and moving obstacles. Obstacles as small as a curb may trip a fast-moving vehicle, so detecting small objects is of great importance. Overhangs and trees may look like large obstacles at a distance, but traveling underneath is often possible. Thus, obstacle detection must consider the 3-D geometry of the world. Figure 5 depicts a typical output of the obstacle detection routine in an urban environment. Each red object corresponds to an obstacle. Towards the bottom right, a camera image is shown for reference.

The robot's primary sensor for obstacle detection is the Velodyne laser. A simple algorithm for detecting obstacles in Velodyne scans would be to find points with similar x-y coordinates whose vertical displacement exceeds a given threshold. Indeed, this algorithm can be used to detect large obstacles such as pedestrians, signposts, and cars. However, range and calibration error are high enough with this sensor that the displacement threshold cannot be set low enough in practice to detect curb-sized objects without substantial numbers of false positives.

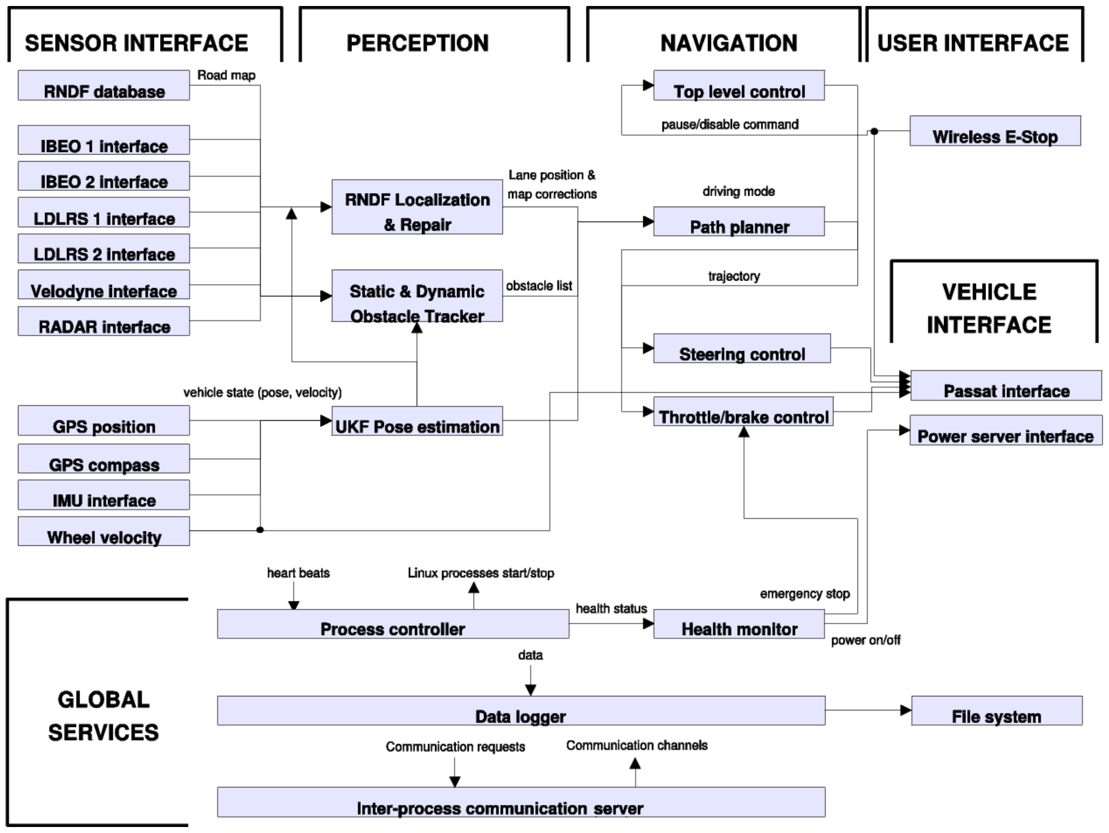


Figure 3: Flow diagram of the Junior Software.

An alternative to comparing vertical displacements is to compare the range returned by two adjacent beams, where "adjacency" is measured in terms of the pointing angle of the beams. Each of the 64 lasers has a fixed pitch angle relative to the vehicle frame, and thus would sweep out a circle of a fixed radius on a flat ground plane as the sensor rotates. Sloped terrain locally compresses these rings, causing the distance between adjacent rings to be smaller than the inter-ring distance on flat terrain. In the extreme case, a vertical obstacle causes adjacent beams to return nearly equal ranges. Because the individual beams strike the ground at such shallow angles, the distance between rings is a much more sensitive measurement of terrain slope than vertical displacement. By finding points that generate inter-ring distances that differ from the expected distance by more than a given threshold, even obstacles that are not apparent to the vertical thresholding algorithm can be reliably detected.

In addition to terrain slope, rolling and pitching of the vehicle will cause the rings traced out by the individual lasers to compress and expand. If this is not taken into account, rolling to the left can cause otherwise flat terrain to the left of the vehicle to be detected incorrectly as an obstacle. This problem can be remedied by making the expected distance to the next ring a function of range, rather than the index of the particular laser. Thus as the vehicle rolls to the left, the expected range difference for a specific beam decreases as the ring moves closer to the vehicle. Implemented in this way, small obstacles can be reliably detected even as the sensor rolls and pitches.

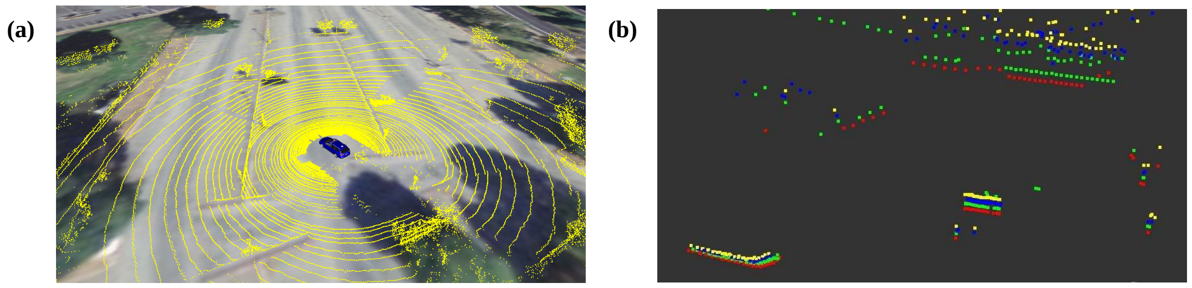


Figure 4: (a) The Velodyne contains 64 laser sensors and rotates at 10 Hz. It is able to see objects and terrain out to 60 meters in every direction. (b) The IBEO sensor possesses four scan lines which are primarily parallel to the ground. The IBEO is capable of detecting large vertical obstacles, such as cars and signposts.

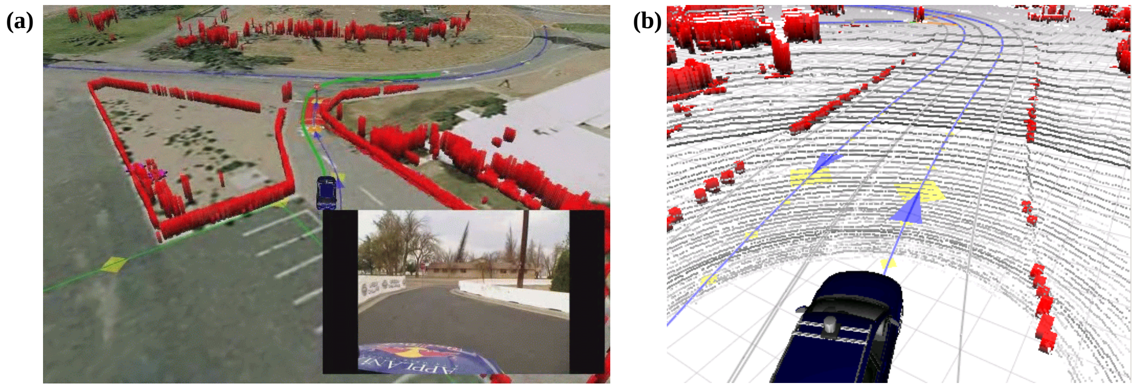


Figure 5: Obstacles detected by the vehicle are overlayed over aerial imagery (left) and Velodyne data (right). In the example on the right, the curbs along both sides of the road are detected.

Two more issues must be addressed when performing obstacle detection in urban terrain. First, trees and other objects frequently overhang safe driving surfaces and should not be detected as obstacles. Overhanging objects are filtered out by comparing their height with a simple ground model. Points that fall in a particular x-y grid cell that exceed the height of the lowest detected point in the same cell by more than a given threshold (the height of the vehicle plus a safety buffer), are ignored as overhanging obstacles.

Second, the Velodyne sensor possesses a “blind spot” behind the vehicle. This is the result of the sensor’s geometry and mounting location. Further, it also cannot detect small obstacles such as curbs in the immediate vicinity of the robot due to self-occlusion. Here the IBEO and SICK LDLRS sensors are used to supplement the Velodyne data. Because both of these sensors are essentially 2-D, ground readings cannot be distinguished from vertical obstacles, and hence obstacles can only be found at very short range (where ground measurements are unlikely). Whenever either of these sensors detects an object within a close range (15 meters for the LDLRS and 5 meters for the IBEO), the measurement is flagged as an obstacle. This combination between short-range sensing in 2-D and longer range sensing using the 3-D sensor provides high reliability. We note that a 5 meter cut-off for the IBEO sensor may seem overly pessimistic, as this laser is designed for long range detection (100 meters and more). However, the sensor presents a large number of

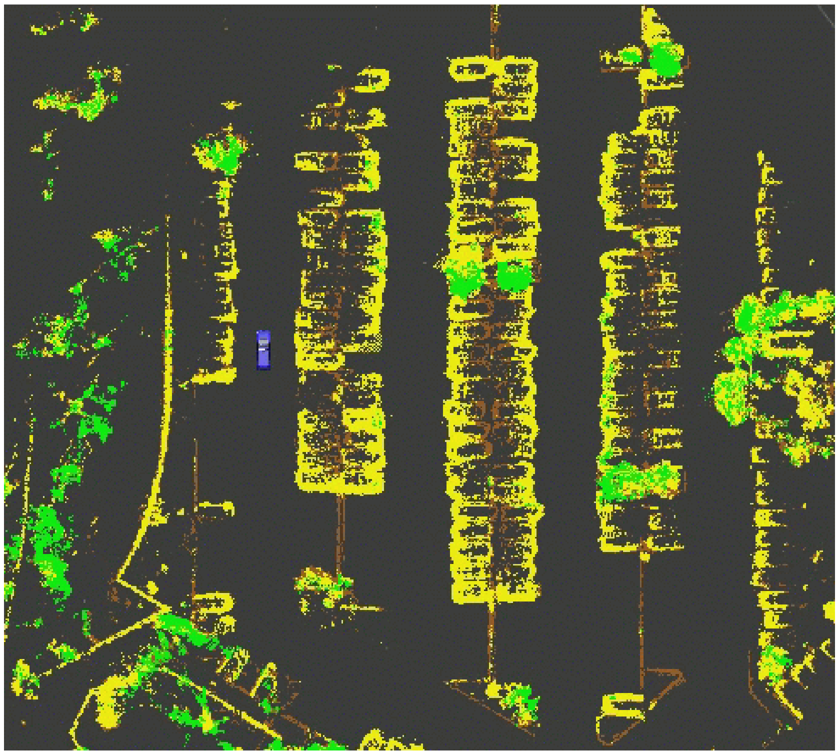


Figure 6: A map of a parking lot. Obstacles colored in yellow are tall obstacles, brown obstacles are curbs, and green obstacles are overhanging objects (e.g. tree branches) that are of no relevance to ground navigation.

false positive detections on non-flat terrain, such as dirt roads.

Our obstacle detection method worked exceptionally well. In the Urban Challenge, we know of no instance in which our robot Junior collided with an obstacle. In particular, Junior never ran over a curb. We also found that the number of false positives was remarkably small, and false positives did not measurably impact the vehicle performance. In this sense, static obstacle detection worked flawlessly.

4.2 Static Mapping

In many situations, multiple measurements have to be integrated over time even for static environment mapping. Such is the case, for example, in parking lots, where occlusion or range limitations may make it impossible to see all relevant obstacles at all times. Integrating multiple measurements is also necessary to cope with certain blind spots in the near range of the vehicle. In particular, curbs are only detectable beyond a certain minimum range with a Velodyne laser. To alleviate these problems, Junior caches sensor measurement into local maps. Figure 6 shows such a local map, constructed from many sensor measurements over time. Different colors indicate different obstacle types on a parking lot. The exact map update rule relies on the standard Bayesian framework for evidence accumulation (Moravec, 1988). This safeguards the robot against spurious obstacles that only show up in a small number of measurements.

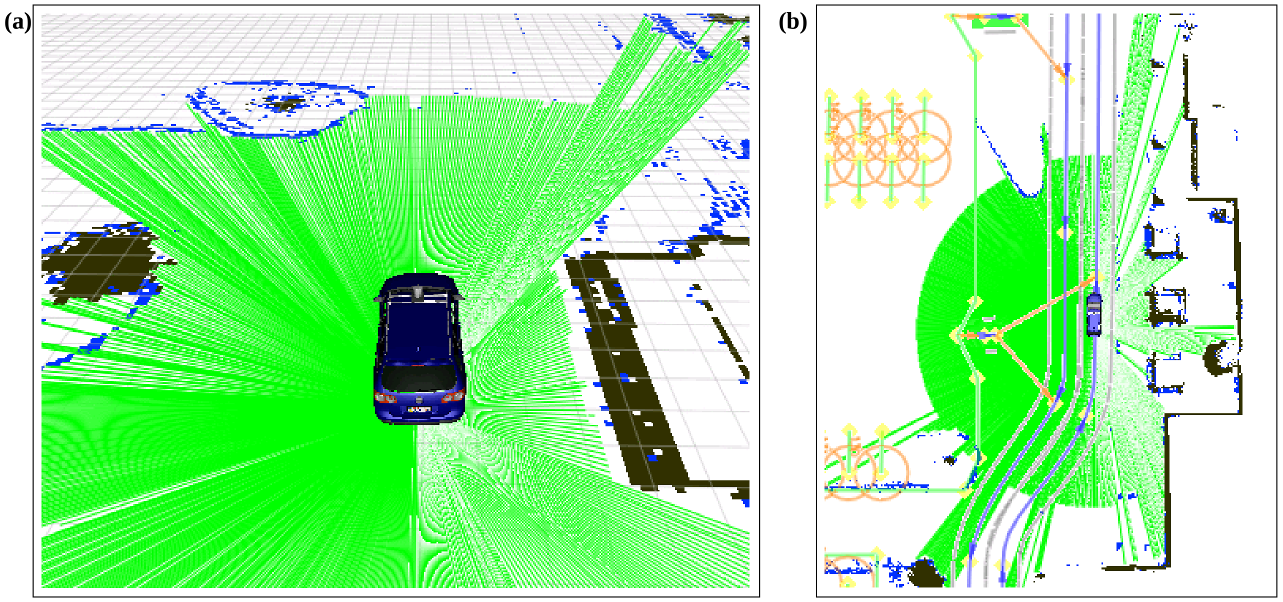


Figure 7: Examples of free space analysis for Velodyne scans. The green lines represent the area surrounding the robot that is observed to be empty. This evidence is incorporated into the static map, shown in black and blue.

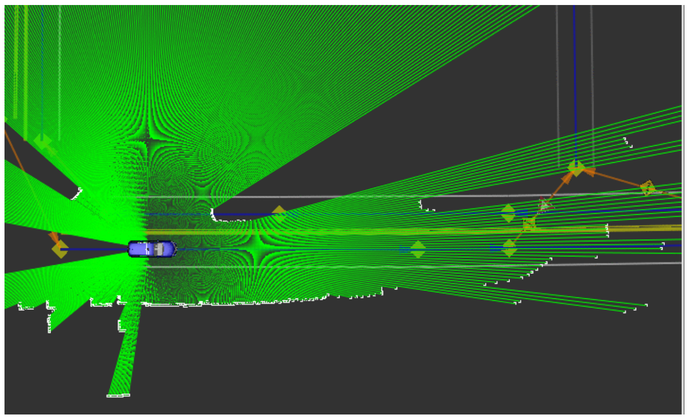
A key downside of accumulating static data over time into a map arises from objects that move. For example, a passage may be blocked for a while, and then become drivable again. To accommodate such situations, the software performs a local visibility calculation. In each polar direction away from the robot, the grid cells between the robot and the nearest detected object are observed to be free. Beyond the first detected obstacle, of course, it is impossible to say whether the absence of further obstacles is due to occlusion. Hence, no map updating takes place beyond this range. This mechanism may still lead to an overly conservative map, but empirically works well for navigating cluttered spaces such as parking lots. Figure 7 illustrates the region in which free space is detected in a Velodyne sensor scan.

4.3 Dynamic Object Detection and Tracking

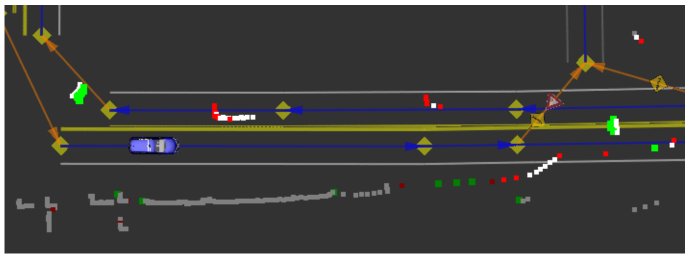
A key challenge in successful urban driving pertains to other moving traffic. The present software provides a reliable method for moving object detection and prediction based on particle filters.

Moving object detection is performed on a synthetic 2-D scan of the environment. This scan is synthesized from the various laser sensors by extracting the range to the nearest detected obstacle along an evenly spaced array of synthetic range sensors. The use of such a synthetic scan comes with several advantages over the raw sensor data. First, its compactness allows for efficient computation. Second, the method is applicable to any of the three obstacle-detecting range sensors (Velodyne, IBEO, and SICK LDLRS), and any combination thereof. The latter property stems from the fact that any of those laser measurements can be mapped easily into a synthetic 2-D range scan, rendering the scan representation relatively sensor-independent. This synergy thus provides our robot with a unified method for finding, tracking, and predicting moving objects. Figure 8a

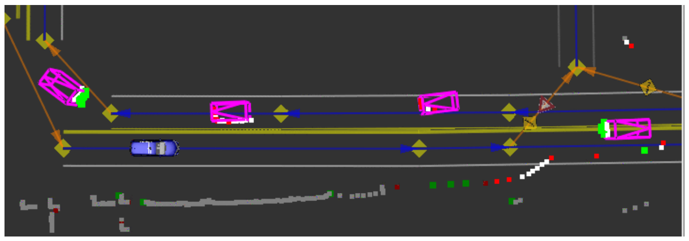
(a)



(b)



(c)



(d)

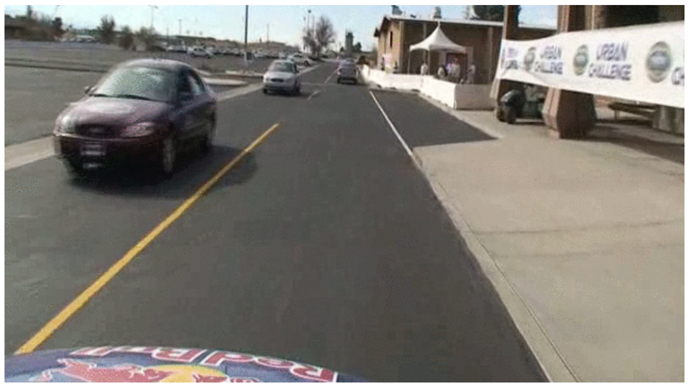


Figure 8: (a) Synthetic 2-D scan derived from Velodyne data. (b) Scan differencing provides areas in which change has occurred, colored here in green and red. (c) Tracks of other vehicles. (d) The corresponding camera image.

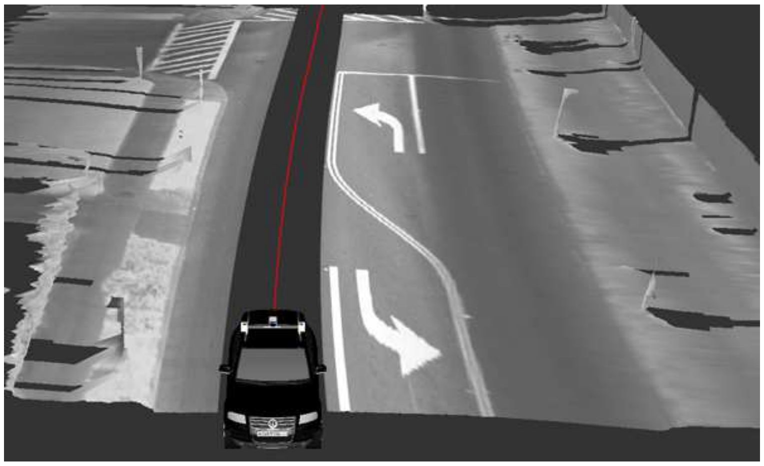


Figure 9: The side lasers provide intensity information that is matched probabilistically with the RNDF for precision localization.

shows such a synthetic scan.

The moving object tracker then proceeds in two stages. First, it identifies *areas of change.* For that, it compares two synthetic scans acquired over a brief time interval. If an obstacle in one of the scans falls into the free space of the respective other scan, this obstacle is a witness of motion. Figure 8b shows such a situation. The red color of a scan corresponds to an obstacle that is new, and the green color marks the absence of a previously seen obstacle.

When such witnesses are found, the tracker initializes a set of particles as possible object hypotheses. These particles implement rectangular objects of different dimensions, and at slightly different velocities and locations. A particle filter algorithm is then used to track such moving objects over time. Typically, within three sightings of a moving object, the filter latches on and reliably tracks the moving object.

Figure 8c depicts the resulting tracks; a camera image of the same scene is shown in Figure 8d. The tracker estimates the location, the yaw, the velocity, and the size of the object.

5 Precision Localization

One of the key perceptual routines in Junior's software pertains to localization. As noted, the robot is given a digital map of the road network in form of an RNDF. While the RNDF is specified in GPS coordinates, the GPS-based inertial position computed by the Applanix system is generally not able to recover the coordinates of the vehicle with sufficient accuracy to perform reliable lane keeping without sensor feedback. Further, the RNDF is itself inaccurate, adding further errors if the vehicle were to blindly follow the road using the RNDF and Applanix pose estimates. Junior therefore estimates a local alignment between the RNDF and its present position using local sensor measurements. In other words, Junior continuously localizes itself relative to the RNDF.

This fine-grained localization uses two types of information: road reflectivity and curb-like obsta-

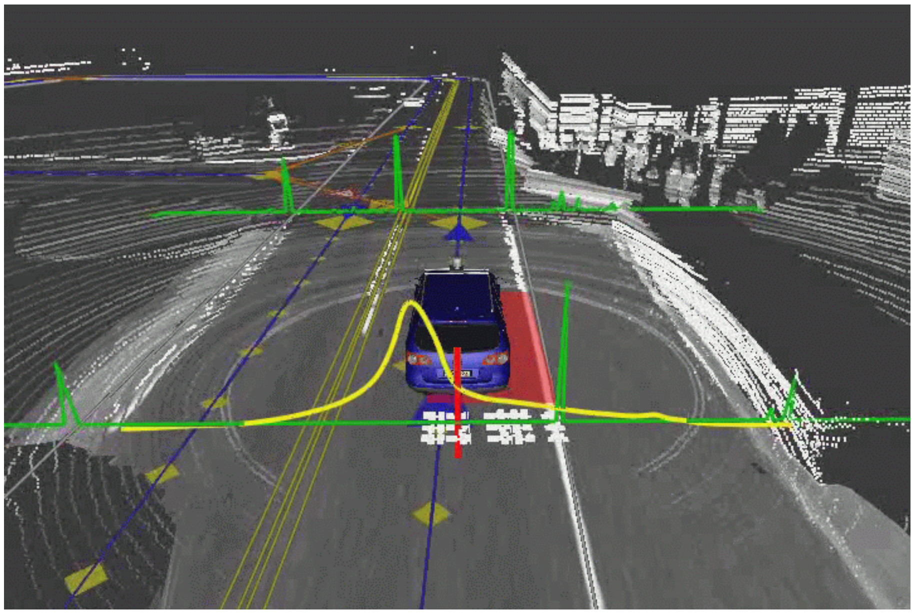


Figure 10: Typical localization result: The red bar illustrates the Applanix localization, whereas the yellow curve measures the posterior over the lateral position of the vehicle. The green line depicts the response from the lane line detector. In this case, the error is approximately 80 cm.

cles. The reflectivity is sensed using the RIEGL LMS-Q120 and the SICK LMS sensors, both of which are pointed towards the ground. Fig. 9 shows the reflectivity information obtained through the sideways mounted SICK sensors, and integrated over time. This diagram illustrates the varying infrared reflectivity of the lane markings.

The filter for localization is a 1-D histogram filter which estimates the vehicle's lateral offset relative to the RNDF. This filter estimates the posterior distribution of any lateral offset based on the reflectivity and the sighted curbs along the road. It "rewards," in a probabilistic fashion, offsets for which lane-marker-like reflectivity patterns align with the lane markers or the road side in the RNDF. The filter "penalizes" offsets for which an observed curb would reach into the driving corridor of the RNDF. As a result, at any point in time the vehicle estimates a fine-grained offset to the measured location by the GPS-based INS system.

Figure 10 illustrates localization relative to the RNDF in a test run. Here the green curves depicts the likely locations of lane markers in both lasers, and the yellow curve depicts the posterior distribution in the lateral direction. This specific posterior deviates from the Applanix estimate by about 80 cm, which, if not accounted for, would make Junior's wheels drive on the center line. In the Urban Challenge Event, localization offsets of 1 meter or more were common. Without this localization step, Junior would have frequently crossed the center line unintentionally, or possibly hit a curb.

Finally, Figure 11 shows a distribution of lateral offset corrections that were applied during the Urban Challenge.

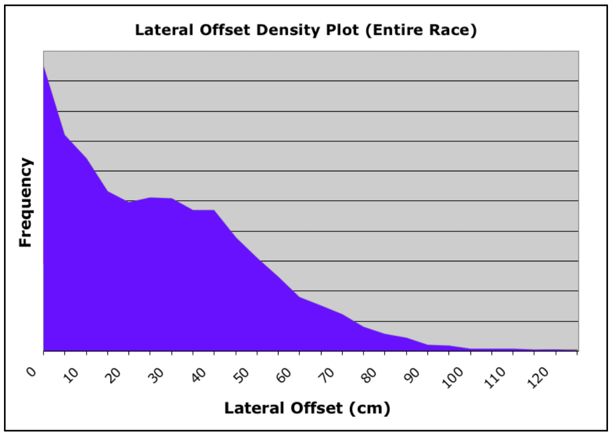


Figure 11: Histogram of average localization corrections during the race. At times the lateral correction exceeds one meter.

5.1 Smooth Coordinates

When integrating multiple sensor measurements over time, it may be tempting to use the INS pose estimates (the output of the Applanix) to calculate the relative offset between different measurements. However, in any precision INS system, the estimated position frequently "jumps" in response to GPS measurements. This is because INS systems provide the *most likely* position at the present time. As new GPS information arrives, it is possible that the most likely position changes by an amount inconsistent with the vehicle motion. The problem, then, is that when such a revision occurs, past INS measurements have to be corrected as well, to yield a consistent map. Such a problem is known in the estimation literature as (backwards) smoothing (Jazwinsky, 1970).

To alleviate this problem, Junior maintains an internal *smooth* coordinate system that is robust to such jumps. In the smooth coordinate system, the robot position is defined as the sum of all incremental velocity updates:

$$\bar{x} \quad = \quad x_0 + \sum_t \Delta t \cdot \dot{x}_t$$

where x_0 is the first INS coordinate, and $\dot{x}_t$ are the velocity estimates of the INS. In this internal coordinate system, sudden INS position jumps have no effect, and the sensor data are always locally consistent. Vehicle velocity estimates from the pose estimation system tend to be much more stable than the position estimates, even when GPS is intermittent or unavailable. X and Y velocities are particularly resistant to jumps because they are partially observed by wheel odometry.

This "trick" of smooth coordinates makes it possible to maintain locally consistent maps even when GPS shifts occur. We note, however, that the smooth coordinate system may cause inconsistencies in mapping data over long time periods, hence can only be applied to local mapping problems. This is not a problem for the present application, as the robot only maintains local maps for navigation.

In the software implementation, the mapping between raw (global) and smooth (local) coordinates only requires that one maintain the sum of all estimation shifts, which is initialized by zero. This correction term is then recursively updated by adding mismatches between actual INS coordinates and the velocity-based value.

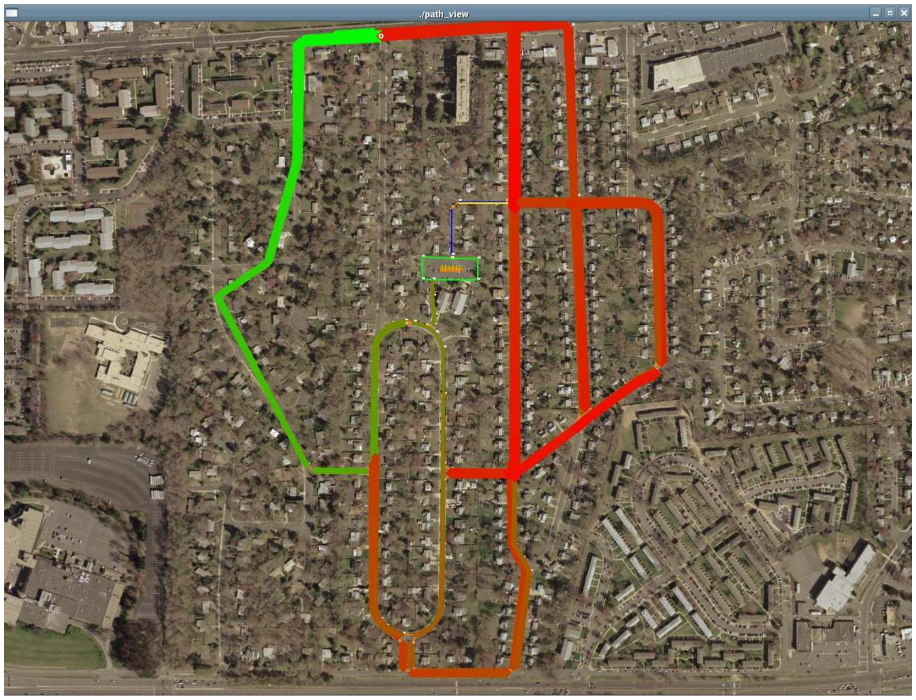


Figure 12: Global planning: Dynamic programming propagates values through a crude discrete version of the environment map. The color of the RNDF is representative of the cost to move to the goal from each position in the graph. Low costs are green and high costs are red.

6 Navigation

6.1 Global Path Planning

The first step of navigation pertains to global path planning. The global path planner is activated for each new checkpoint; it also is activated when a permanent road blockage leads to a change of the topology of the road network. However, instead of planning one specific path to the next checkpoint, the global path planner plans paths from every location in the map to the next checkpoint. As a result, the vehicle may depart from the optimal path and select a different one without losing direction as to where to move.

Junior's global path planner is an instance of *dynamic programming*, or DP (Howard, 1960). The DP algorithm recursively computes for each cell in a discrete version of the RNDF the *cumulative costs* of moving from each such location to the goal point. The recursive update equation for the cost is standard in the DP literature. Let $V(x)$ be the cost of a discrete location in the RNDF, with $V(\mathrm{goal}) = 0$. Then the following recursive equation defines the backup and, implicitly, the cumulative cost function V:

$$V(x) \quad \longleftarrow \quad \min_u \; c(x,u) + \sum_y p(y \mid x,u) \; V(y)$$

Here u is an action, e.g., drive along a specific road segment. In most cases, there is only one admissible action. At intersections, however, there are choices (go straight, turn left, ...). Multi-lane roads offer the choice of lane changes. For these cases the maximization over the control choice u in the expression above will provide multiple terms, the minimization of which leads to the fastest expected path.

In practice, not all action choices are always successful. For example, a shift from a left to a right lane only "succeeds" if there is no vehicle in the right lane; otherwise the vehicle cannot shift lanes. This is accommodated in the use of the transition probability $p(y \mid x, u)$. Junior, for example, might assess the success probability of a lane shift at any given discrete location as low as 10%. The benefit of this probabilistic view of decision making is that it penalizes plans that delay lane changes to the very last moment. In fact, Junior tends to execute lane shifts at the earliest possibility, and it trades off speed gains with the probability (and the cost) of failure when passing a slow moving vehicle at locations where a subsequent right turn is required (which may only be admissible when in the right lane).

A key ingredient in the recursive equation above is the cost $c(x, u)$. In most cases, the cost is simply the time it takes to move between adjacent cells in the discrete version of the RNDF. In this way, the speed limits are factored into the optimal path calculation, and the vehicle selects the path that in expectation minimizes arrival time. Certain maneuvers, such as left turns across traffic, are "penalized" by an additional time penalty to account for the risk that the robot takes when making such a choice. In this way, the cost function c implements a careful balance between navigation time and risk. So in some cases, Junior engages on a slight detour so as to avoid a risky left turn, or a risky merge. The additional costs of maneuvers can either be set by hand (as they were for the Urban Challenge) or learned from simulation data in representative environments.

Figure 12 shows a propagated cumulative cost function. Here the cumulative cost is indicated by the color of the path. This global function is brought to bear to assess the "goodness" of each location beyond the immediate sensor reach of the vehicle.

6.2 RNDF Road Navigation

The actual vehicle navigation is handled differently for common road navigation and the free-style navigation necessary for parking lots.

Figure 13 visualizes a typical situation. For each principal path, the planner rolls out a trajectory that is parallel to the smoothed center of the lane. This smoothed lane center is directly computed from the RNDF. However, the planner also rolls out trajectories that undergo lateral shifts. Each of those trajectories is the result of an internal vehicle simulation with different steering parameters. The score of a trajectory considers the time it will take to follow this path (which may be infinite if a path is blocked by an obstacle), plus the cumulative cost computed by the global path planner, for the final point along the trajectory. The planner then selects the trajectory which minimizes this total cost value. In doing so, the robot combines optimal route selection with dynamic nudging around local obstacles.

Figure 14 illustrates this decision process in a situation where a slow-moving vehicle blocks the right lane. Even though lane changes come with a small penalty cost, the time savings due to faster travel on the left lane result in a lane change. The planner then steers the robot back into the right lane when the passing maneuver is complete.

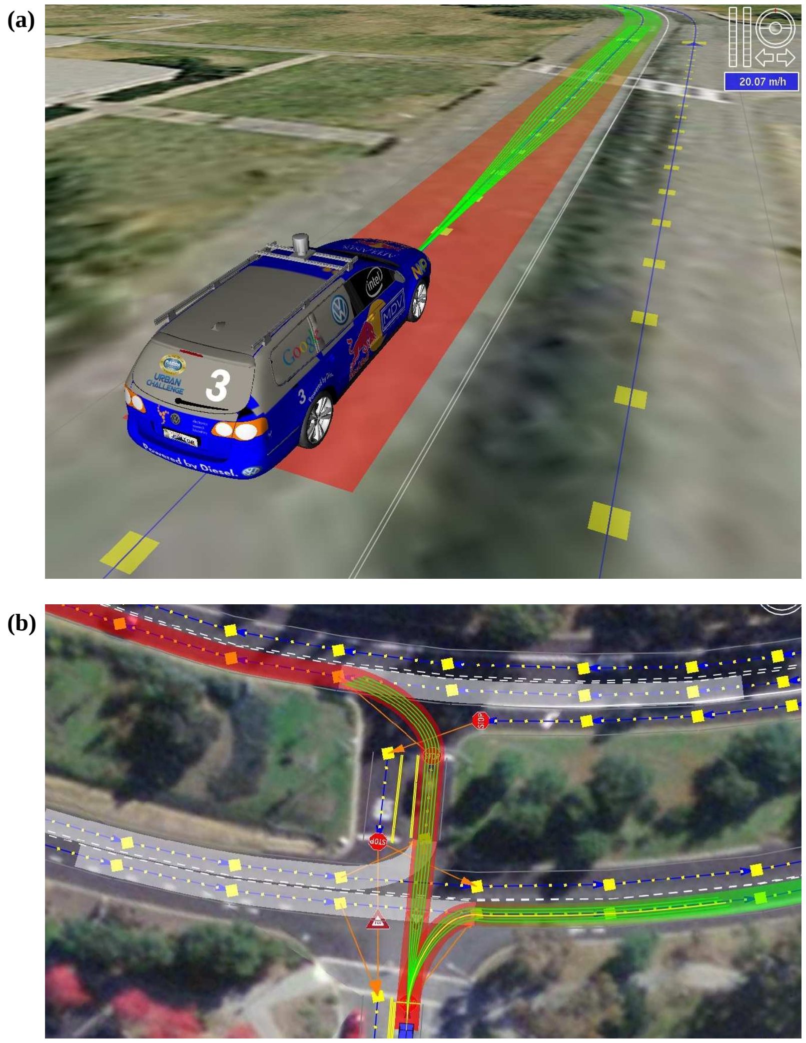


Figure 13: Planner roll-outs in an urban setting with multiple discrete choices. (a) For each principle path, the planner rolls out trajectories that undergo lateral shifts. (b) A driving situation with two discrete plan choices, turn right or drive straight through the intersetion. The paths are colored according to the DP value function, with red being high cost and green being low cost.

We find that this path planner works well in well-defined traffic situations. It results in smooth motion along unobstructed roads, and in smooth and well-defined passing maneuvers. The planner also enables Junior to avoid small obstacles that might extend into a lane, such as parked cars on the side. However, it is unable to handle blocked roads or intersections, and it also is unable to navigate parking lots.

6.3 Free-Form Navigation

For free-form navigation in parking lots, the robot utilizes a second planner, which can generate arbitrary trajectories irrespective of a specific road structure. This planner requires a goal coordinate and a map. It identifies a near-cost optimal path to the goal should such a path exist.

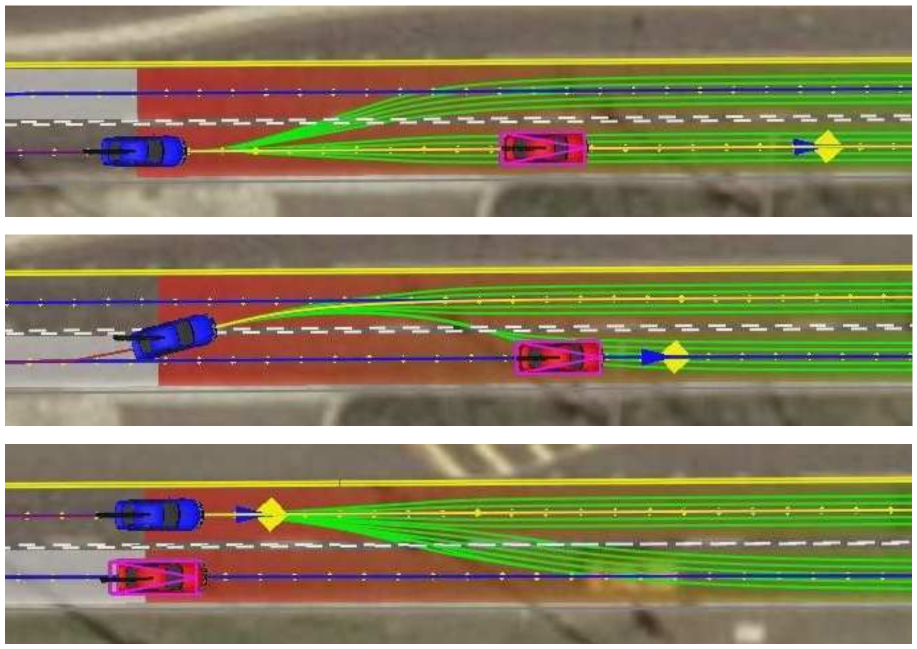


Figure 14: A passing maneuver. The additional cost of being in a slightly sub-optimal lane is overwhelmed by the cost of driving behind a slow driver, causing Junior to change lanes and pass.

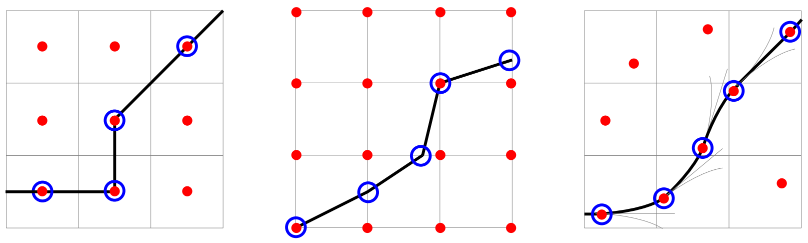


Figure 15: Graphical comparison of search algorithms. Left: A* associates costs with centers of cells and only visits states that correspond to grid-cell centers. Center: Field D* (Ferguson and Stentz, 2005) associates costs with cell corners and allows arbitrary linear paths from cell to cell. Right: Hybrid A* associates a continuous state with each cell and the score of the cell is the cost of its associated continuous state.

This free-form planner is a modified version of A*, which we call *hybrid A**. In the present application, hybrid A* represents the vehicle state in a 4-D discrete grid. Two of those dimensions represent the x-y-location of the vehicle center in smooth map coordinates; a third the vehicle heading direction θ, and a forth dimension pertains the direction of motion, either forward or reverse.

One problem with regular (non-hybrid) A* is that the resulting discrete plan cannot be executed by a vehicle, simply because the world is continuous, whereas A* states are discrete. To remedy this problem, hybrid A* assigns to each discrete cell in A* a continuous vehicle coordinate. This continuous coordinate is such that it can be realized by the actual robot.

To see how this works, let $\langle x, y, \theta \rangle$ be the present coordinates of the robot, and suppose those

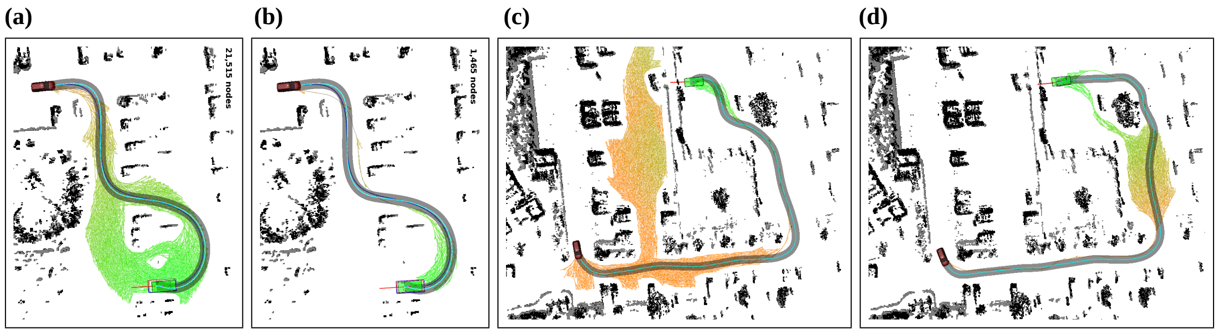


Figure 16: Hybrid-state A* heuristics. (a) Euclidean distance in 2-D expands $21,515$ nodes. (b) The non-holonomic-without-obstacles heuristic is a significant improvement, as it expands $1,465$ nodes, but as shown in (c), it can lead to wasteful exploration of dead-ends in more complex settings ($68,730$ nodes). (d) This is rectified by using the latter in conjunction with the holonomic-with-obstacles heuristic ($10,588$ nodes).

coordinates lie in cell c_i in the discrete A* state representation. Then, by definition, the continuous coordinates associated with cell c_i are $x_i = x$, $y_i = y$, and $\theta_i = \theta$. Now predict the (continuous) vehicle state after applying a control u for a given amount of time. Suppose the prediction is $\langle x', y', \theta' \rangle$, and assume this prediction falls into a different cell, denoted c_j. Then, if this is the first time c_j has been expanded, this cell will be assigned the associated continuous coordinates $x_j = x'$, $y_j = y'$, and $\theta_j = \theta'$. The result of this assignment is that there exists an actual control u in which the continuous coordinates associated with cell c_j can actually be attained—a guarantee which is not available for conventional A*. The hybrid A* algorithm then applies the same logic for future cell expansions, using $\langle x_j, y_j, \theta_j \rangle$ whenever making a prediction that starts in cell c_j. We note that hybrid A* is guaranteed to yield realizable paths, but it is not complete. That is, it may fail to find a path. The coarser the discretization, the more often hybrid A* will fail to find a path.

Figure 15 compares hybrid A* to regular A* and Field D* (Ferguson and Stentz, 2005), an alternative algorithm that also considers the continuous nature of the underlying state space. A path found by plain A* cannot easily be executed; and even the much smoother Field D* path possesses kinks that a vehicle cannot execute. By virtue of associating continuous coordinates with each grid cell in Hybrid A*, our approach results in a path that is executable.

The cost function in A* follows the idea of execution time. Our implementation assigns a slightly higher cost to reverse driving to encourage the vehicle to drive “normally.” Further, a change of direction induces an additional cost to account for the time it takes to execute such a maneuver. Finally, we add a pseudo-cost that relates to the distance to nearby obstacles so as to encourage the vehicle to stay clear of obstacles.

Our search algorithm is guided by two heuristics, called the *non-holonomic-without-obstacles heuristic* and the *holonomic-with-obstacles heuristic*. As the name suggests, the first heuristic ignores obstacles but takes into account the non-holonomic nature of the car. This heuristic, which can be completely pre-computed for the entire 4D space (vehicle location, and orientation, and direction of motion), helps in the end-game by approaching the goal with the desired heading. The second heuristic is a dual of the first in that it ignores the non-holonomic nature of the car, but computes the shortest distance to the goal. It is calculated online by performing dynamic programming

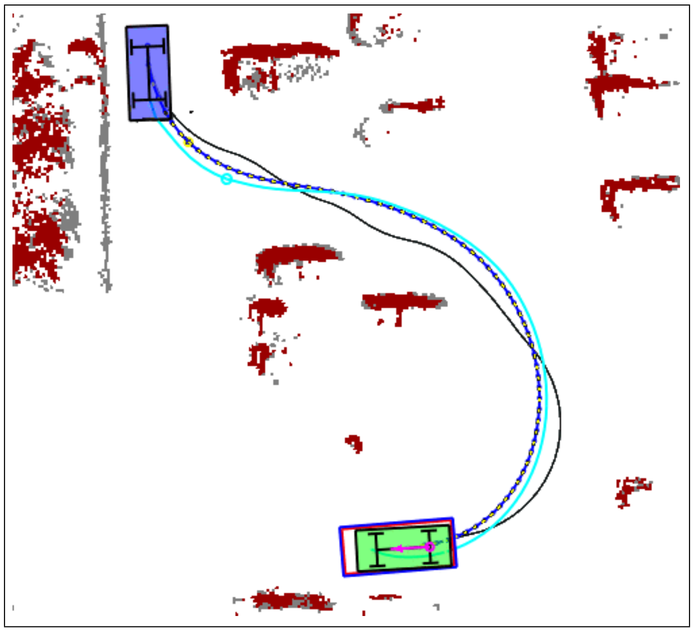


Figure 17: Path smoothing with Conjugate Gradient. This smoother uses a vehicle model to guarantee that the resulting paths are attainable. The Hybrid A* path is shown in black. The smoothed path is shown in blue (front axle) and cyan (rear axle). The optimized path is much smoother than the Hybrid A* path, and can thus be driven faster.

in 2-D (ignoring vehicle orientation and motion direction). Both heuristics are admissible, so the maximum of the two can be used.

Figure 16a illustrates A* planning using the commonly used Euclidean distance heuristic. As shown in Figure 16b, the non-holonomic-without-obstacles heuristic is significantly more efficient than Euclidean distance, since it takes into account vehicle orientation. However, as shown in Figure 16c, this heuristic alone fails in situations with U-shaped dead ends. By adding the holonomic-with-obstacles heuristic, the resulting planner is highly efficient, as illustrated in Figure 16d.

While hybrid A* paths are realizable by the vehicle, the small number of discrete actions available to the planner often lead to trajectories with rapid changes in steering angles, which may still lead to trajectories that require excessive steering. In a final post-processing stage, the path is further smoothed by a Conjugate Gradient smoother that optimizes similar criteria as hybrid A*. This smoother modifies controls and moves waypoints locally. In the optimization, we also optimize for minimal steering wheel motion and minimum curvature. Figure 17 shows the result of smoothing.

The hybrid A* planner is used for parking lots and also for certain traffic maneuvers, such as U-turns. Figure 18 shows examples from the Urban Challenge and the associated National Qualification Event. Shown there are two successful U-turns and one parking maneuver. The example in Figure 18d is based on a simulation of a more complex parking lot. The apparent suboptimality of the path is the result of the fact that the robot “discovers” the map as it explores the environment, forcing it into multiple backups as a previously believed free path is found to be occupied. All of those runs involve repetitive executions of the hybrid A* algorithm, which take place while the

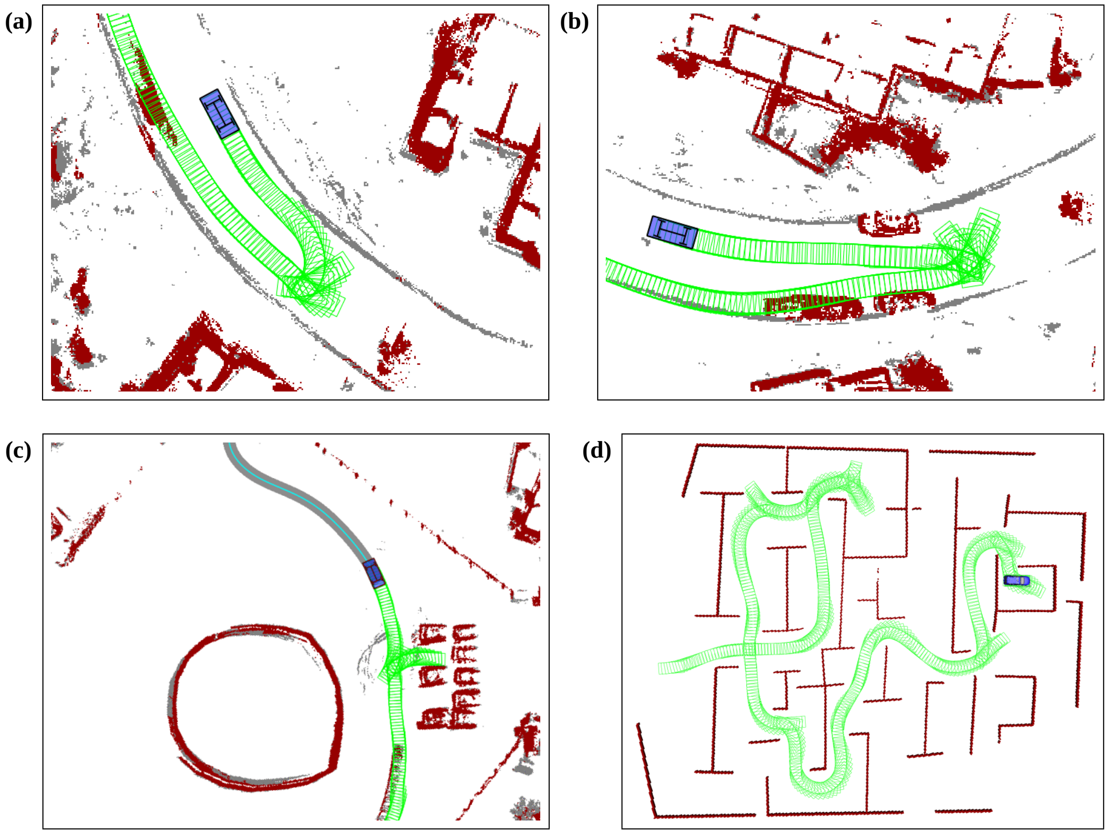


Figure 18: Examples of trajectories generated by Junior's hybrid A* planner. Trajectories in (a)–(c) were driven by Junior in the DARPA Urban challenge: (a),(b) show U-turns on blocked roads, (c) shows a parking task. The path in (d) was generated in simulation for a more complex maze-like environment. Note that in all cases the robot had to replan in response to obstacles being detected by its sensors. In particular, this explains the sub-optimality of the trajectory in (d).

vehicle is in motion. When executed on a single core of Junior's computers, planning from scratch requires up to 100 milliseconds; in the Urban Challenge, planning was substantially faster because of the lack of obstacles in parking lots.

6.4 Intersections and Merges

Intersections are places that require discrete choices not covered by the basic navigation modules. For example, at multi-way intersections with stop signs, vehicles may only proceed through the intersection in the order of their arrival.

Junior keeps track of specific "critical zones" at intersections. For multi-way intersections with stop signs, such critical zones correspond to regions near each stop sign. If such a zone is occupied by a vehicle at the time the robot arrives, Junior waits until this zone has cleared (or a timeout has occurred). Intersection critical zones are shown in Figure 19. In merging, the critical zones correspond to segments of roads where Junior may have to give precedence to moving traffic. If an object is found in such a zone, Junior uses its radars and its vehicle tracker to determine the

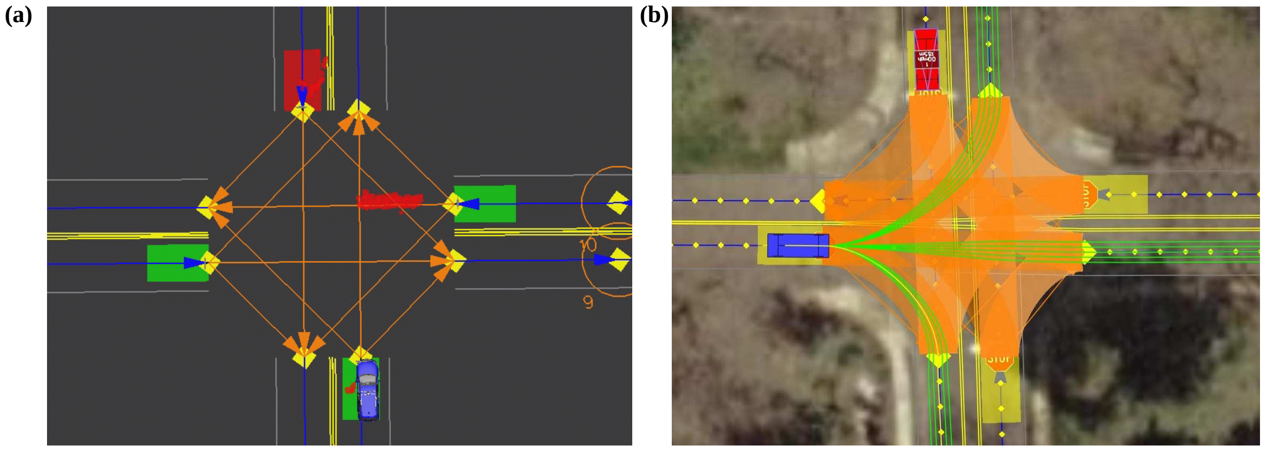


Figure 19: Critical zones: (a) At this four-way stop sign, busy critical zones are colored in red, whereas critical zones without vehicles are shown in green. In this image, a vehicle can be seen driving through the intersection from the right. (b) Critical zones for merging into an intersection.

velocity of moving objects. Based on the velocity and proximity, a threshold test then marks the zone in question as busy, which then results in Junior waiting at a merge point. The calculation of critical zones is somewhat involved. However, all computations are performed automatically based on the RNDF, and ahead of the actual vehicle operation.

Figure 20 visualizes a merging process during the qualification event to the Urban Challenge. This test involves merging into a busy lane with 4 human-driven vehicles, and across another lane with 7 human-driven cars. The robot waits until none of the critical zones are busy, and then pulls into the moving traffic. In this example, the vehicle was able to pull safely into 8 second gaps in two-way traffic.

6.5 Behavior Hierarchy

An essential aspect of the control software is logic that prevents the robot from getting stuck. Junior's *stuckness detector* is triggered in two ways: through timeouts when the vehicle is waiting for an impasse to clear, and through the repeated traversal of a location in the map—which may indicate that the vehicle is looping indefinitely.

Figure 21 shows the finite state machine (FSM) that is used to switch between different driving states, and that invokes exceptions to overcome stuckness. This FSM possesses 13 states (of which 11 are shown; 2 are omitted for clarity). The individual states in this FSM correspond to the following conditions:

- LOCATE_VEHICLE: This is the initial state of the vehicle. Before it can start driving, the robot estimates its initial position on the RNDF, and starts road driving or parking lot navigation, whichever is appropriate.
- FORWARD_DRIVE: This state corresponds to forward driving, lane keeping and obstacle avoidance. When not in a parking lot, this is the preferred navigation state.

(a)

(b)

(c)

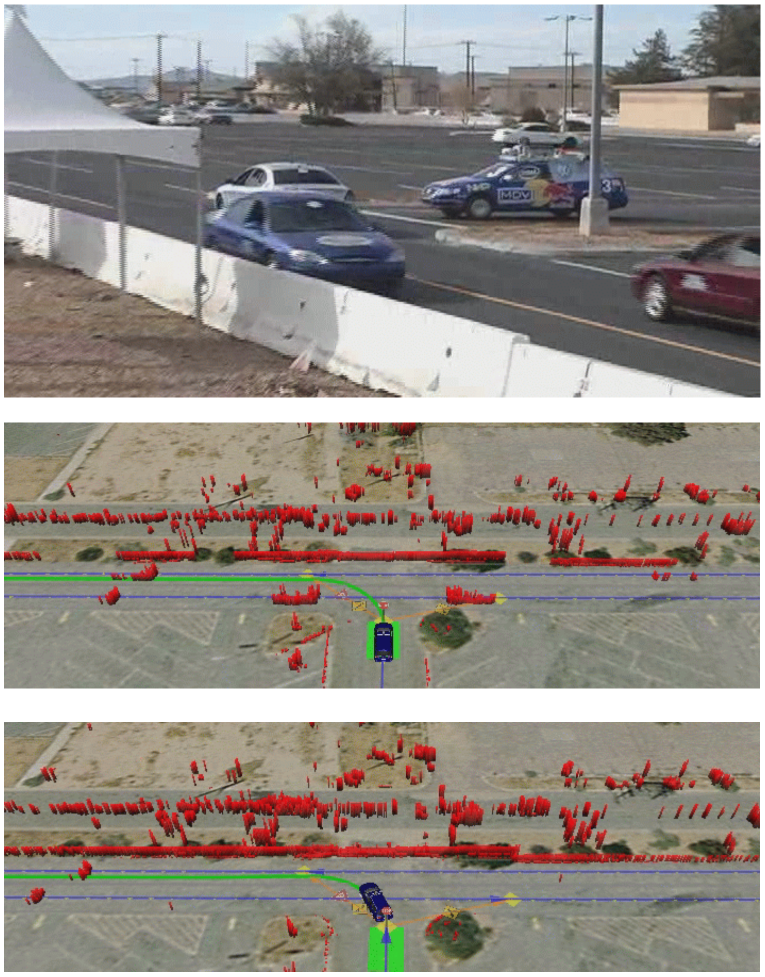


Figure 20: Merging into dense traffic during the qualification events at the Urban Challenge. (a) Photo of merging test; (b)-(c) The merging process.

- STOP_SIGN_WAIT: This state is invoked when the robot waits at at a stop sign to handle intersection precedence.
- CROSS_INTERSECTION: Here the robot waits if it is safe to cross an intersection (e.g., during merging), or until the intersection is clear (if it is an all-way stop intersection). The state also handles driving until Junior has exited the intersection.
- STOP_FOR_CHEATERS: This state enables Junior to wait for another car moving out of turn at a four way intersection.
- UTURN_DRIVE: This state is invoked for a U-turn.
- UTURN_STOP: Same as UTURN_DRIVE, but here the robot is stopping in preparation for a U-turn.
- CROSS_DIVIDER: This state enables Junior to cross the yellow line (after stopping and waiting for oncoming traffic) in order to avoid a partial road blockage.
- PARKING_NAVIGATE: Normal parking lot driving.
- TRAFFIC_JAM: In this sate, the robot uses the general-purpose hybrid A* planner to get

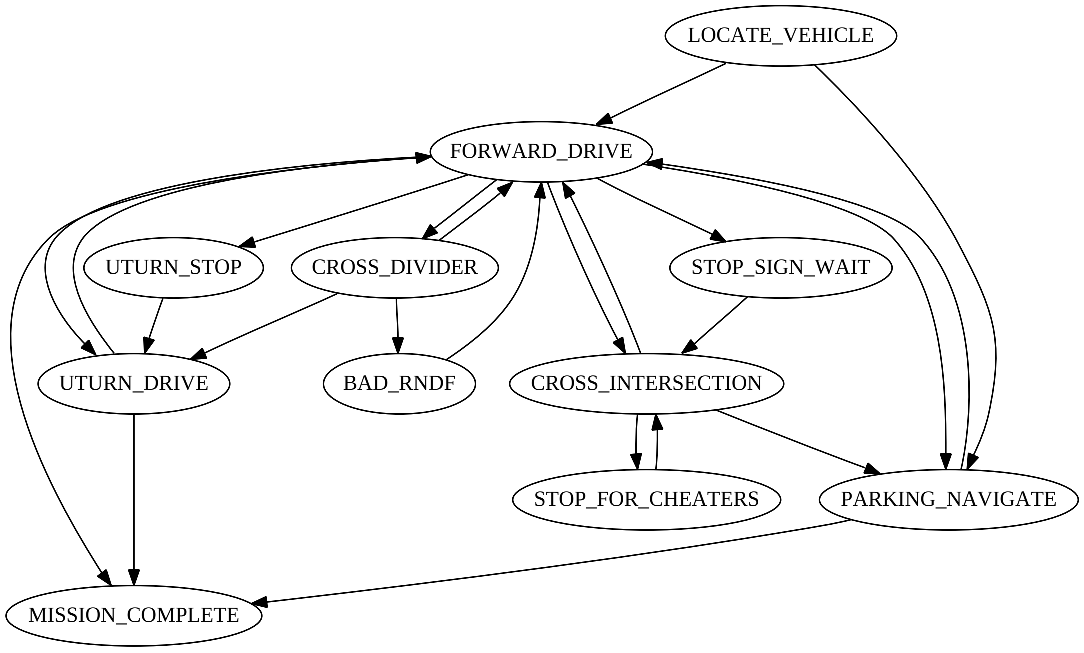


Figure 21: Finite State Machine that governs the robot's behavior.

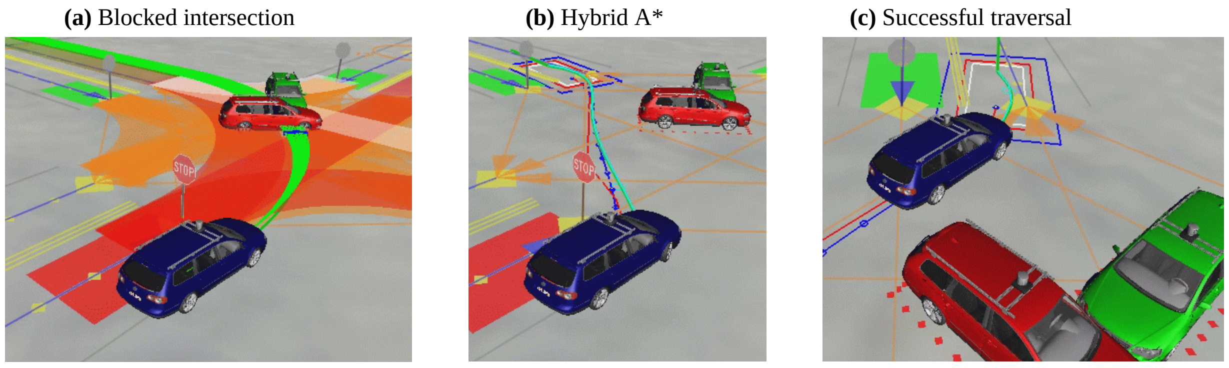


Figure 22: Navigating a simulated traffic jam: After a timeout period, the robot resorts to hybrid A* to find a feasible path across the intersection.

around a road blockage. The planner aims to achieve any road point 20 meters away on the current robot trajectory. Use of the general-purpose planner allows the robot to engage in unrestricted motion and disregard certain traffic rules.

- ESCAPE: This state is the same as TRAFFIC_JAM, only more extreme. Here the robot aims for any waypoint on any base trajectory more than 20 meters away. This state enables the robot to choose a suboptimal route at an intersection in order to extract itself out of a jam.
- BAD_RNDF: In this state, the robot uses the hybrid A* planner to navigate a road that does not match the RNDF. It triggers on one lane, one way roads if CROSS_DIVIDER fails.
- MISSION_COMPLETE: This state is set when race is over.

For simplicity, Figure 21 omits ESCAPE and TRAFFIC_JAM. Nearly all states have transitions to ESCAPE and TRAFFIC_JAM.

At the top level, the FSM transitions between the normal driving states, such as lane keeping and parking lot navigation. Transitions to lower driving levels (exceptions) are initiated by the stuckness detectors. Most of those transition invoke a “wait period” before the corresponding exception behavior is invoked. The FSM returns to normal behavior after the successful execution of a robotic behavior.

The FSM makes the robot robust to a number of contingencies. For example:

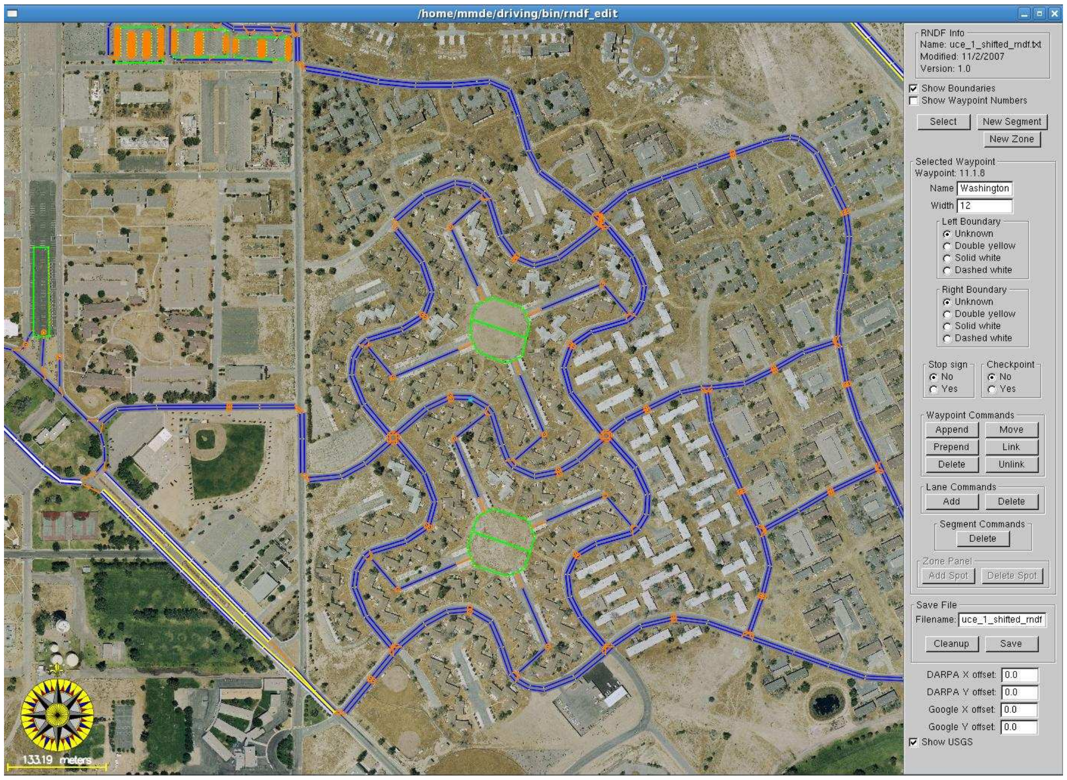
- For a blocked lane, the vehicle considers crossing into the opposite lane. If the opposite lane is also blocked, a U-turn is initiated, the internal RNDF is modified accordingly, and dynamic programming is run to regenerate the RNDF value function.
- Failure to traverse a blocked intersection is resolved by invoking the hybrid A* algorithm, to find a path to the nearest reachable exit of the intersection; see Figure 22 for an example.
- Failure to navigate a blocked one-way road results in using hybrid A* to the next GPS waypoint. This feature enables vehicles to navigate RNDFs with sparse GPS waypoints.
- Repeated looping while attempting to reach a checkpoint results in the checkpoint being skipped, so as to not jeopardize the overall mission. This behavior avoids infinite looping if a checkpoint is unreachable.
- Failure to find a path in a parking lot with hybrid A* makes the robot temporarily erase its map. Such failures may be the result of treating as static objects that since moved away – which cannot be excluded.
- In nearly all situations, failure to make progress for extended periods of time ultimately leads to the use of hybrid A* to find a path to a nearby GPS waypoint. When this rare behavior is invoked, the robot does not obey traffic rules any longer.

In the Urban Challenge event, the robot almost never entered any of the exception states. This is largely because the race organizers repeatedly paused the robot when it was facing traffic jams. However, extensive experiments prior to the Urban Challenge showed that it was quite difficult to make the robot fail to achieve its mission, provided that the mission remained achievable.

6.6 Manual RNDF Adjustment

Ahead of the Urban Challenge event, DARPA provided teams not just with an RNDF, but also with a high-resolution aerial image of the site. While the RNDF was produced by careful ground-based GPS measurements along the course, the aerial image was purchased from a commercial vendor and acquired by aircraft.

To maximize the accuracy of the RNDF, the team manually adjusted and augmented the DARPA-provided RNDF. Figure 23 shows a screen shot of the editor. This tool enables an editor to move, add, and delete waypoints. The RNDF editor program is fast enough to incorporate new waypoints in real time (10Hz).



RNDF Info
Name: uce_1_shifted_rndf.txt
Modified: 11/2/2007
Version: 1.0

[✓ Show Boundaries]
Show Waypoint Numbers

Selected Waypoint
Waypoint: 11.1.8
Name: Washington
Width: 12

Left Boundary: [✓ Unknown] Double yellow Solid white Dashed white

Right Boundary: [✓ Unknown] Double yellow Solid white Dashed white

Stop sign: [✓ No] Yes

Checkpoint: [✓ No] Yes

Save File
Filename: uce_1_shifted_rndf

DARPA X offset: 0.0
DARPA Y offset: 0.0
Google X offset: 0.0
Google Y offset: 0.0

[✓ Show USGS]

Figure 23: RNDF editor tool.

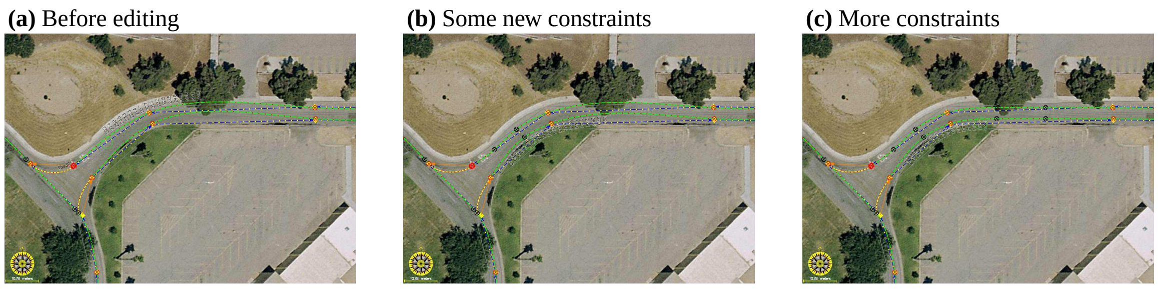


Figure 24: Example: Effect of adding and moving waypoints in the RNDF. Here the corridor is slightly altered to better match the aerial image. The RNDF editor permits for such alterations in an interactive manner, and displays the results on the base trajectory without any delay.

The editing required three hours of a person's time. In an initial phase, waypoints were shifted manually, and roughly 400 new way points were added manually to the 629 lane waypoints in the RNDF. Those additions increased the spatial coherence of the RNDF and the aerial image. Figure 24 shows a situation in which the addition of such additional waypoint constraints leads to substantial improvements of the RNDF.

To avoid sharp turns at the transition of linear road segments, the tool provides an automated RNDF smoothing algorithm. This algorithm upsamples the RNDF at one meter intervals, and sets those as to maximize the smoothness of the resulting path. The optimization of these additional points combines a least squares distance measure with a smoothness measure. The resulting "smooth

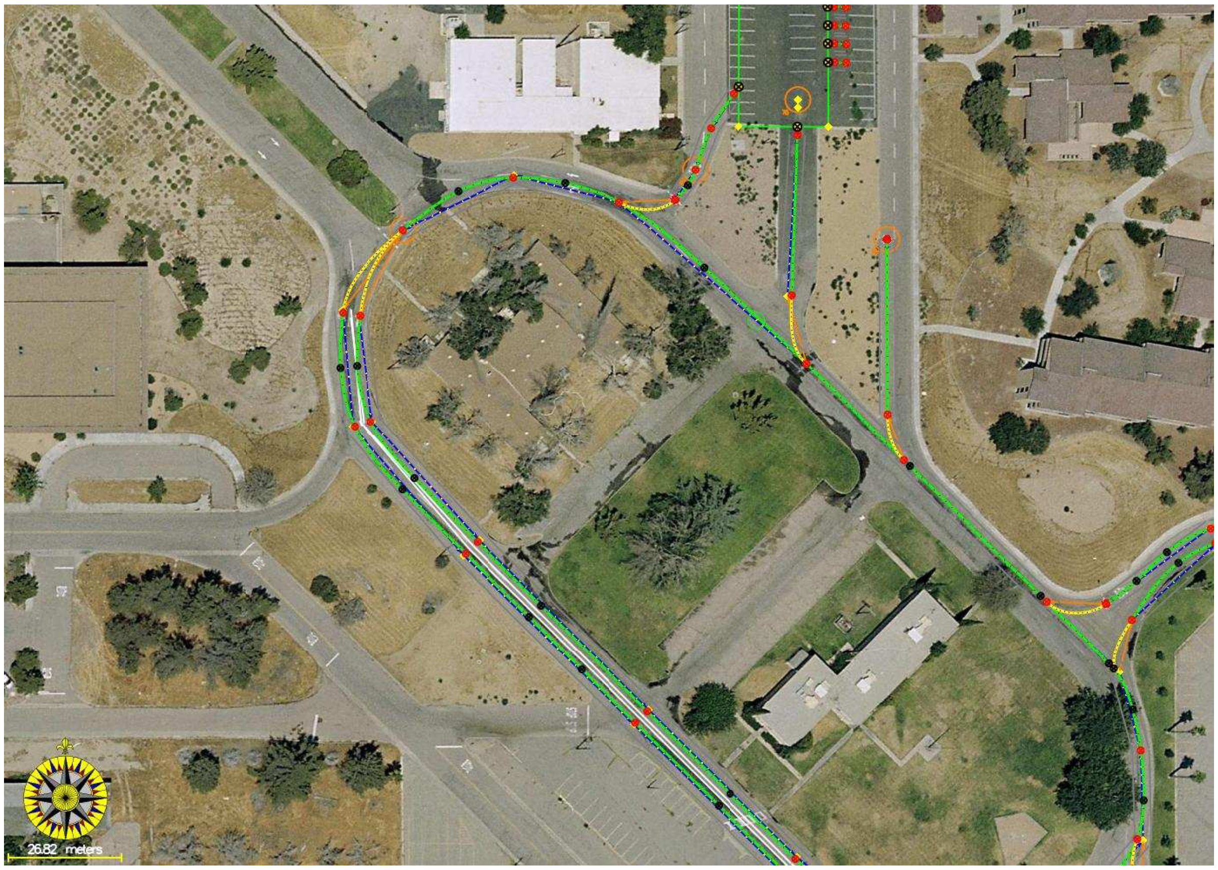


Figure 25: The SRNDF creator produces a smooth base trajectory automatically by minimizing a set of nonlinear quadratic constraints. The original RNDF is shown in blue. The smooth SRNDF is shown in green.

RNDF," or SRNDF, is then used instead of the original RNDF for localization and navigation. Figure 25 compares the RNDF and the SRNDF for a small fraction of the course.

7 The Urban Challenge

7.1 Results

The Urban Challenge took place Nov. 3, 2007, in Victorville, CA. Figure 26 shows images of the start and the finish of the Urban Challenge. Our robot Junior never hit an obstacle, and according to DARPA, it broke no traffic rule. A careful analysis of the race logs and official DARPA documentation revealed two situations (described below) in which Junior behaved suboptimally. However, all of those events were deemed rule conforming by the race organizers. Overall, Junior's localization and road following behaviors were essentially flawless. The robot never came close to hitting a curb or crossing into opposing traffic.

The event was organized in three missions, which differed in length and complexity. Our robot accomplished all three missions in 4 hours 5 minutes, and 6 seconds of run time. During this time,

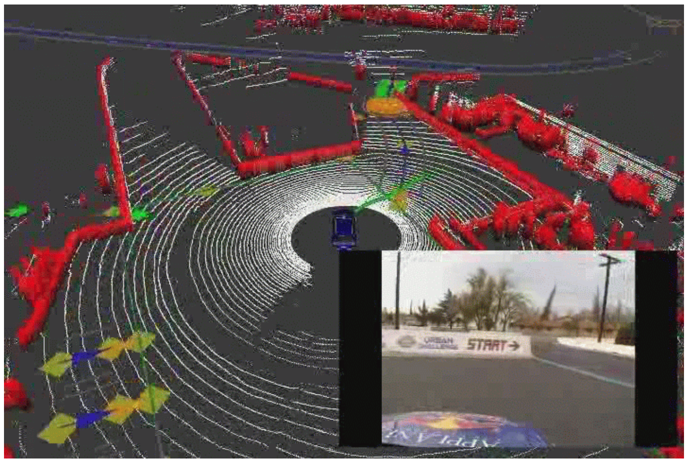


Figure 26: The start and the finish of the Urban Challenge. Junior arrives at the finish line.

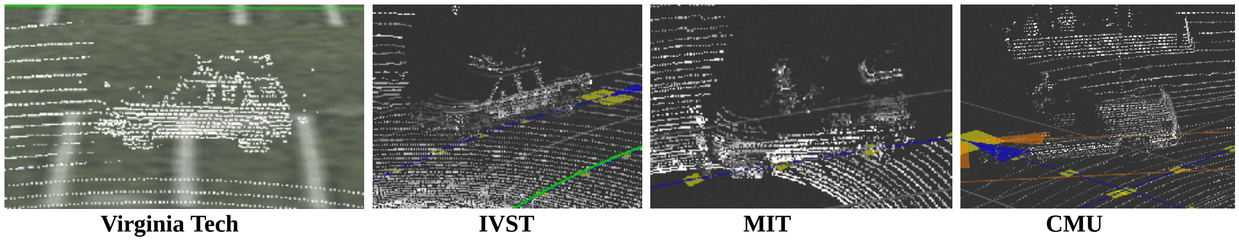


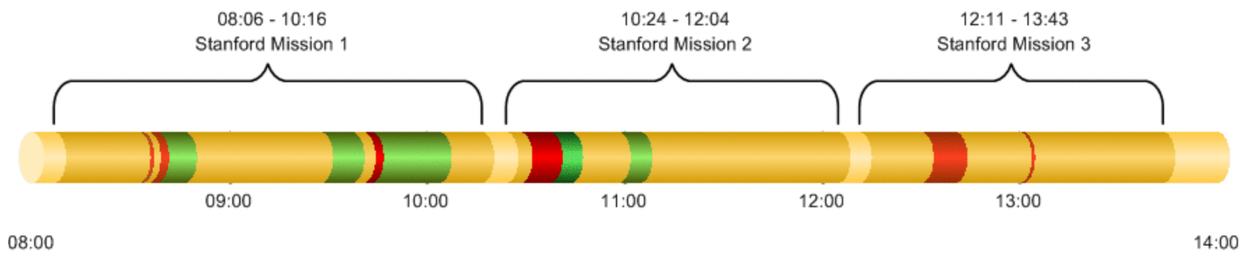
Figure 27: Scans of other robots encountered in the race.

the robot traveled a total of 55.96 miles, or 90.068 km. Its average speed while in run mode was thus 13.7 mph. This is slower than the average speed in the 2005 Grand Challenge (Montemerlo et al., 2006; Urmson et al., 2004), but most of the slowdown was caused by speed limits, traffic regulations (e.g., stop signs), and other traffic. The total time from the start to the final arrival was 5 hours, 23 minutes, and 2 seconds, which includes all pause times. Thus, Junior was paused for a total of 1 hour, 17 minutes and 56 seconds. None of those pauses were caused by Junior, or requested by our team. An estimated 26 minutes and 27 seconds were "local" pauses, in which Junior was paused by the organizers because other vehicles were stuck. Our robot was paused six times because other robots encountered problems on the off-road section, or were involved in an accident. The longest local pause (10 min, 15 sec) occurred when Junior had to wait behind a two-robot accident. Because of DARPA's decision to pause robots, Junior could not exercise its hybrid A* planner in these situations. DARPA determined Junior's adjusted total time to be 4 hours, 29 minutes, and 28 seconds. Junior was judged to be the second fastest finishing robot in this event.

7.2 Notable Race Events

Figure 27 shows scans of other robots encountered in the race. Overall, DARPA officials estimate that Junior faced approximately 200 other vehicles during the race. The large number of robot-robot encounters was a unique feature of the Urban Challenge.

There were several notable encounters during the race in which Junior exhibited particularly intel-



Pause Type	Why	When	Duration
Local	Little Ben: Problems on dirt road	08:32:23 - 08:33:52	0:01:08
Local	Little Ben: Did not leave the dirt road	08:35:28 - 08:37:58	0:02:30
All	Little Ben: Rescue team repositioned the car on Phantom East	08:37:58 - 08:46:30	0:08:32
All	AnnieWay, IVS, Terramax: car got removed	09:27:52 - 09:37:20	0:09:28
Local	Little Ben: Accident with UCF	09:40:42 – 09:43:12	0:02:30
All	Little Ben: Rescue Team	09:43:12 - 10:03:50	0:20:38
Local	Cornell: Stopped without reason, moved on, stopped, ...	10:28:26 - 10:37:30	0:09:04
All	UCF: car got removed	10:37:30 – 10:43:44	0:06:14
All	Carolo: Accident with MIT, car got removed	10:58:11 - 11:04:46	0:06:35
Local	Cornell: Accident with MIT	12:30:11 - 12:40:27	0:10:15
Local	Little Ben: using same parking area	13:00:00 - 13:01:00	0:01:00

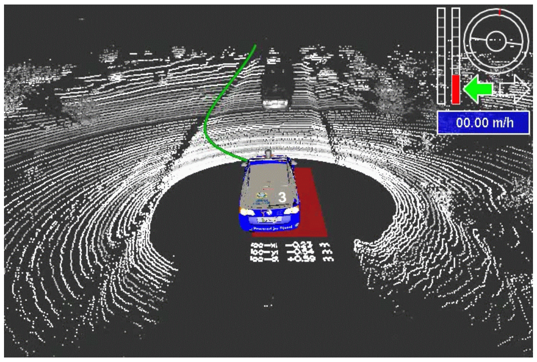
Figure 28: Junior mission times during the Urban Challenge. Times marked green correspond to local pauses, and times in red to all-pauses, in which all vehicles were paused.

ligent driving behavior, as well as two incidents where Junior made clearly suboptimal decisions (neither of which violated any traffic rules).

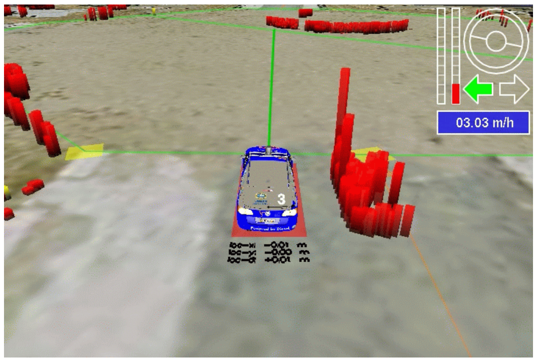
Hybrid A* on the Dirt Road

While the majority of the course was paved, urban terrain, the robots were required to traverse a short off-road section connecting the urban road network to a 30mph highway section. The off-road terrain was graded dirt path with a non-trivial elevation change, reminiscent of the 2005 DARPA Grand Challenge course. This section caused problems for several of the robots in the competition. Junior traveled down the dirt road during the first mission, immediately behind another robot and its chase car. While Junior had no difficulty following the dirt road, the robot in front of Junior stopped three times for extended periods of time. In response to the first stop, Junior also stopped and waited behind the robot and its chase car. After seeing no movement for a period of time, Junior activated several of its recovery behaviors. First, Junior considered CROSS_DIVIDER, a preset passing maneuver to the left of the two stopped cars. There was not sufficient space to fit between the cars and the berm on the side of the road, so Junior then switched to the BAD_RNDF behavior, in which the Hybrid A* planner is used to plan an arbitrary path to the next DARPA waypoint. Unfortunately, there was not enough space to get around the cars even with the general path planner. Junior repeatedly repositioned himself on the road in an attempt to find a free path to the next waypoint, until the cars started moving again. Junior repeated this behavior when the preceding robot stopped a second time, but was paused by DARPA until the first robot recovered. Figure 29a shows data and a CROSS_DIVIDER path around the preceding vehicle on the dirt road.

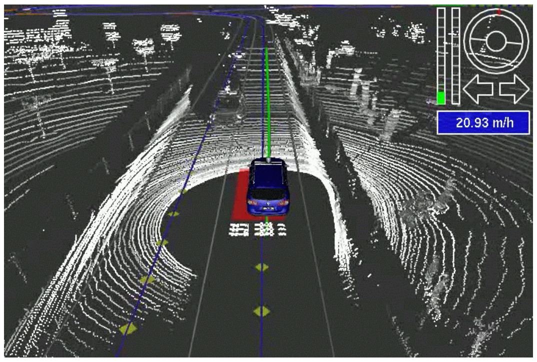
(a) Navigating a blocked dirt road



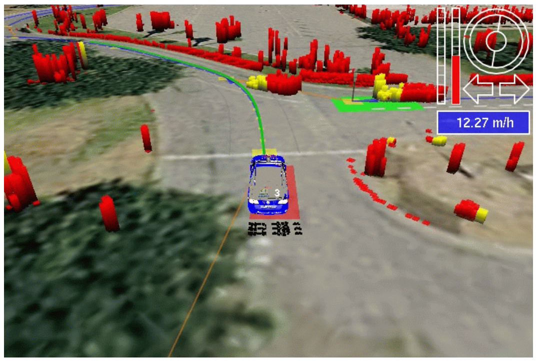
(b) Passing a disabled robot at parking lot entrance



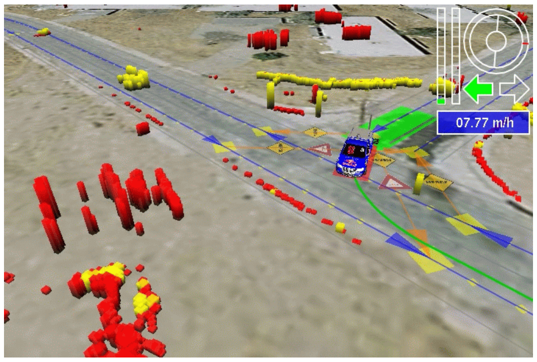
(c) Nudge to avoid an oncoming robot



(d) Slowing down after being cut off by other robot



(e) An overly aggressive merge into moving traffic



(f) Pulling alongside a car at a stop sign

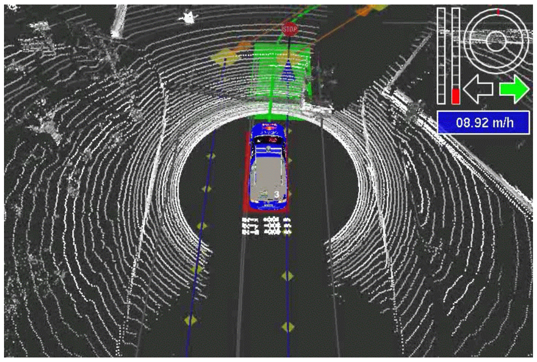


Figure 29: Key moments in the Urban Challenge race.

Passing Disabled Robot

The course included several free-form navigation zones where the robots were required to navigate around arbitrary obstacles and park in parking spots. As Junior approached one of these zones during the first mission, it encountered another robot which had become disabled at the entrance to the zone. Junior queued up behind the robot, waiting for it to enter the zone. After the robot did not move for a given amount of time, Junior passed it slowly on the left using the CROSS_DIVIDER behavior. Once Junior had cleared the disabled vehicle, the Hybrid A* planner was enabled to navigate successfully through the zone. Figure 29b shows this passing maneuver.

Avoiding Opposing Traffic

During the first mission, Junior was traveling down a two-way road and encountered another robot in the opposing lane of traffic. The other robot was driving such that its left wheels were approximately one foot over the yellow line, protruding into oncoming traffic. Junior sensed the oncoming vehicle and quickly nudged the right side of its lane, where it then passed at full speed without incident. This situation is depicted in Figure 29c.

Reacting to an Aggressive Merge

During the third mission, Junior was traveling around a large traffic circle which featured prominently in the competition. Another robot was stopped at a stop sign waiting to enter the traffic circle. The other robot pulled out aggressively in front of Junior, who was traveling approximately 15mph at the time. Junior braked hard to slow down for the other robot, and continued with its mission. Figure 29d depicts the situation during this merge.

Junior Merges Aggressively

Junior merged into moving traffic successfully on numerous occasions during the race. On one occasion during the first mission, however, Junior turned left from a stop sign in front of a robot that was moving at 20mph with an uncomfortably small gap. Data from this merge is shown in Figure 29e. The merge was aggressive enough that the chase car drivers paused the other vehicle. Later analysis revealed that Junior saw the oncoming vehicle, yet believed there was a sufficient distance to merge safely. Our team had previously lowered merging distance thresholds to compensate for overly conservative behavior during the qualification event. In retrospect, these thresholds were set too low for higher speed merging situations. While this merge was definitely suboptimal behavior, it was later judged not be a violation of the rules by DARPA.

Pulling Alongside a Waiting Car

During the second mission, Junior pulled up behind a robot waiting at a stop sign. The lane was quite wide, and the other robot was offset towards the right side of the lane. Junior, on the other hand, was traveling down the left side of the lane. When pulling forward, Junior did not register the other car as being inside the lane of travel, and thus began to pull alongside of the car waiting at the stop sign. As Junior tried to pass, the other car pulled forward from the stop sign and left the area. This incident highlights how difficult it can be for a robot to distinguish between a car stopped at a stop sign and a car parked on the side of the road. See Figure 29f.

8 Discussion

This paper described a robot designed for urban driving. Stanford's robot Junior integrates a number of recent innovations in mobile robotics, such as probabilistic localization, mapping, tracking, global and local planning, and an FSM for making the robot robust to unexpected situations. The results of the Urban Challenge, along with prior experiments carried out by the research team, suggest that the robot is capable of navigating in other robotic and human traffic. The robot successfully demonstrated merging, intersection handling, parking lot navigation, lane changes, and autonomous U-turns.

The approach presented here features a number of innovations, which are well-grounded in past research on autonomous driving and mobile robotics. These innovations include the obstacle/curb detection method, the vehicle tracker, the various motion planners, and the behavioral hierarchy that addresses a broad range of traffic situations. Together, these methods provide for a robust system for urban in-traffic autonomous navigation.

Still, a number of advances are required for truly autonomous urban driving. The present robot is unable to handle traffic lights. No experiments have been performed with a more diverse set of traffic participants, such as bicycles and pedestrians. Finally, DARPA frequently paused robots in the Urban Challenge to clear up traffic jams. In real urban traffic, such interventions are not realistic. It is unclear if the present robot (or other robots in this event!) would have acted sensibly in lasting traffic congestion.

References

Buehler, M., Iagnemma, K., and Singh, S., editors (2006). *The 2005 DARPA Grand Challenge: The Great Robot Race*. Springer, Berlin.

DARPA (2007). Urban challenge rules, revision oct. 27, 2007. See www.darpa.mil/grandchallenge/rules.asp.

Ferguson, D. and Stentz, A. (2005). Field D*: An interpolation-based path planner and replanner. In *Proceedings of the 12th International Symposium of Robotics Research (ISRR'05)*, San Francisco, CA. Springer.

Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. MIT Press and Wiley.

Jazwinsky, A. (1970). *Stochastic Processes and Filtering Theory*. Academic, New York.

Montemerlo, M., Thrun, S., Dahlkamp, H., Stavens, D., and Strohband, S. (2006). Winning the DARPA Grand Challenge with an AI robot. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, Boston, MA. AAAI.

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

Simmons, R. and Apfelbaum, D. (1998). A task description language for robot control. In *Proceedings of the Conference on Intelligent Robots and Systems (IROS)*, Victoria, CA.

Urmson, C., Anhalt, J., Clark, M., Galatali, T., Gonzalez, J., Gowdy, J., Gutierrez, A., Harbaugh, S., Johnson-Roberson, M., Kato, H., Koon, P., Peterson, K., Smith, B., Spiker, S., Tryzelaar, E., and Whittaker, W. (2004). High speed navigation of unrehearsed terrain: Red Team technology for the Grand Challenge 2004. Technical Report CMU-RI-TR-04-37, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

U.S. Department of Transportation, B. o. T. S. (2005). Transportation statistics annual report.