Winning the DARPA Grand Challenge with an AI Robot[*]

Michael Montemerlo, Sebastian Thrun,
Hendrik Dahlkamp, David Stavens
Stanford AI Lab, Stanford University
353 Serra Mall
Stanford, CA 94305-9010
{mmde,thrun,dahlkamp,stavens}@stanford.edu

Sven Strohband
Volkswagen of America, Inc.
Electronics Research Laboratory
4009 Miranda Avenue, Suite 150
Palo Alto, California 94304
sven.strohband@vw.com

Abstract

This paper describes the software architecture of Stanley, an autonomous land vehicle developed for high-speed desert driving without human intervention. The vehicle recently won the DARPA Grand Challenge, a major robotics competition. The article describes the software architecture of the robot, which relied pervasively on state-of-the-art AI technologies, such as machine learning and probabilistic reasoning.

Introduction

The DARPA Grand Challenge was a major robot competition organized by the U.S. government to foster research and development in the area of autonomous driving. The highly popularized event required an autonomous ground robot to traverse a 132-mile course through the Mojave desert in no more than 10 hours. While in 2004, no vehicle traveled more than 5% of the course, five vehicles finished in 2005, four of them within the alloted time.

The DARPA Grand Challenge was in large part a software competition. A skilled human driver would have no difficulty traversing the terrain in a standard SUV. Without a driver, however, the robot has to perform functions normally provided by human drivers: it has to sense, decide, and act. Thus, the methods required to win such a race fall into the realm of autonomous robotics and AI.

This article describes the software architecture of Stanley, Stanford's entry in the DARPA Grand Challenge. The software is capable of acquiring sensor data, constructing internal models of the environment, and making driving decisions at speeds up to 38 mph. Major components of the software are based on machine learning, probabilistic reasoning, and real-time control. Probabilistic methods proved essential in overcoming the measurement noise in various sensors. Machine learning was applied in two different ways: offline, to tune various performance-related parameters, and online, to endow the vehicle with the capability to adapt to the terrain. The "glue" between these modules is a distributed architecture that serves as an effective data pipeline for autonomous robotic driving.

This article provides a comprehensive description of Stanley's software architecture. Towards the end of this article, we discuss results from the 2005 DARPA Grand Challenge.

Vehicle

Stanley (Fig. 1) is based on a diesel-powered Volkswagen Touareg R5. The Touareg has four wheel drive, variable-height air suspension, and automatic, electronic locking differentials. To protect the vehicle from environmental impact, the robot has been outfitted with skid plates and a reinforced front bumper. A custom interface enables direct, electronic actuation of both throttle and brakes. A DC motor attached to the steering column provides electronic steering control. Vehicle data, such as individual wheel speeds and steering angle, are sensed automatically and communicated to the computer system through a CAN bus interface.

Nearly all of the vehicle's sensors are mounted on a custom roof rack (Fig. 1b). Five SICK laser range finders are mounted pointing forward along the driving direction of the vehicle, but at slightly different tilt angles. The roof rack also holds a color camera for long-range road perception. Two forward-pointed antennae of a RADAR system are mounted on both sides of the laser sensor array. Further back, the roof rack holds the primary Global Positioning System (GPS) antenna, two antennae for the GPS compass, and various tracking antennae required by the race organizer. A 6-DOF inertial measurement unit (IMU) is mounted in the trunk.

The computing system is located in the vehicle's trunk, as shown in Fig. 1c. The trunk features a shock-mounted rack that carries an array of six 1.6 GHz Pentium M blade computers, a Gigabit Ethernet switch, and various devices that interface to the physical sensors and the vehicle. It also features a custom-made power system with backup batteries and a switch box that enables the robot to power-cycle individual system components through software. The operating system on all computers is Linux.

Software Pipeline

In autonomous driving, the software has to be reliable, robust to errors, and it has to run in real-time. Our vehicle achieves these objectives through a distributed software architecture reminiscent of the well-known *three layer architecture* (Gat 1998). The architecture pipelines data through a series of layers, transforming sensor data into internal models, abstract plans, and concrete robot controls. The use of pipelining for the data flow minimizes the data processing latency, which

Figure 1: (a) The robot shown at approximately 1:40pm on Oct 8, 2005, as it successfully completes the 132-miles course. (b) View of the car and the roof rack with sensors. (c) The computing system in the vehicle's trunk.

is the time between the acquisition of a sensor measurement and its effect on the vehicle motion.

The overall software system consists of about 30 modules executed in parallel (Fig. 2). The system is broken down into six layers which correspond to the following functions: sensor interface, perception, control, vehicle interface, user interface, and global services.

1. The *sensor interface layer* comprises a number of software modules concerned with receiving and time-stamping all sensor data. The layer receives data from the laser sensors at 75Hz, from the camera at approximately 8 Hz, the GPS and GPS compass at 10Hz, and the IMU at 100Hz. This layer also contains a database server with the course coordinates (RDDF file).

2. The *perception layer* maps sensor data into internal models. A primary module in this layer is the UKF vehicle state estimator, which determines the vehicle's coordinates, orientation, and velocities. Three different mapping modules build 2-D environment models based on lasers, the camera, and the radar system. A road finding module uses the laser-derived maps to find the boundary of a road, so that the vehicle can center itself laterally. Finally, a surface assessment module extracts parameters of the current road for the purpose of determining safe vehicle speeds.

3. The *control layer* is responsible for determine the steering, throttle, and brake response of the vehicle. A key module is the path planner, which sets trajectories in steering- and velocity-space. This trajectory is passed to two reactive controllers, one for the steering control and and for brake and throttle control. Both of those generate low-level control commands for the vehicle.

4. The *vehicle interface layer* serves as the interface to the robot's drive-by-wire system. The control layer also features a top level control module, implemented as a simple finite state automaton. This level determines the general vehicle mode in response to user commands, received through the in-vehicle touch screen or the wireless E-stop.

5. The *user interface layer* comprises the remote E-stop and a touch-screen module for starting up the software.

6. The *global services layer* provides a number of basic services for all other software modules. Naming and communication services are provides through CMU's Inter-Process Communication (IPC) toolkit (Simmons & Apfelbaum 1998). A centralized parameter server module maintains a database of all vehicle parameters and updates them
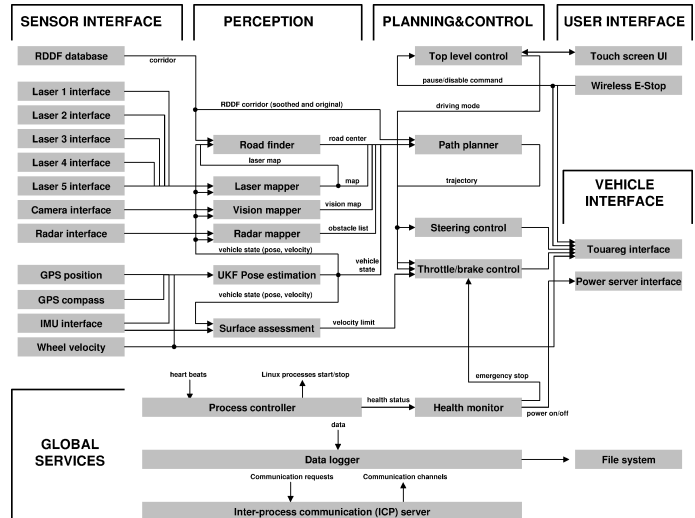

Figure 2: Software flowchart: The software is divided into six functional groups: sensor interface, perception, control, vehicle interface, user interface, and global services.

in a consistent manner. Another module monitors the health of all systems components and restarts processes when necessary. Clock synchronization is achieved thorough a time server. Finally, a data logging server dumps sensor, control, and diagnostic data on disk for replay and analysis.

Key Software Modules

Dynamic State Estimation

The vehicle state is comprised of two groups of variables: parameters characterizing the dynamic state of the vehicle (location, speed, etc), and parameters pertaining to the health of the vehicle. Following the rich literature on vehicle guidance (Grewal, Weill, & Andrews 2001), the dynamic state is comprised of 15 variables: the vehicle coordinates in a global GPS-referenced coordinate frame, the derivatives of these coordinates (vehicle velocity), the orientation of the vehicle relative to the global coordinate frame (in Euler angles, yaw, pitch, and roll), three accelerometer biases, and three gyroscope biases.

An unscented Kalman filter (UKF) (Julier & Uhlmann 1997) estimates these quantities at an update rate of 100Hz.
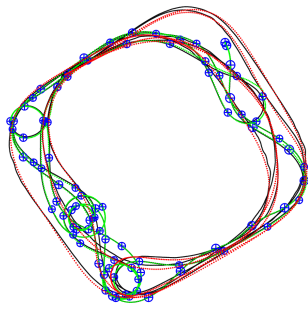
Figure 3: UKF results for a GPS outage that lasts three minutes. The dotted red line is the estimate without GPS (the green line is a portion at which GPS was available). Ground truth is shown in black.
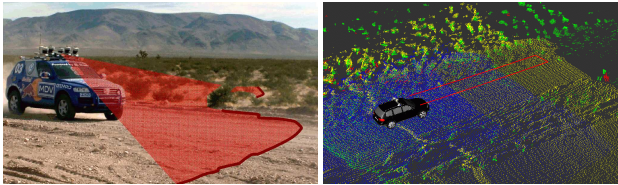

Figure 4: The vehicle uses a single line scanner to acquire surface data from the terrain to make driving decisions. The scan is integrated over time into a 3-D point cloud.

The UKF incorporates observations from the GPS, the GPS compass, the IMU, and the wheel encoders. The GPS system provides both absolute position and velocity measurements, which are both incorporated into the UKF. Thanks to an internal vehicle model, the UKF can survive GPS outages of up to several minutes (see Fig. 3).

Laser Mapping

The lasers are the primary obstacle detection sensors on the vehicle. Fig. 4a illustrates the view of one of the five lasers as it scans the terrain. Each of the lasers is angled slightly downward to scan the terrain at a specific range. The vehicle integrates the laser data over time into a 3-D point cloud, such as the one shown in Fig. 4b.

The robot analyzes this point cloud for pairs of nearby measurements whose z-coordinates differ by an amount large enough to constitute an obstacle. Any pair of such points constitute a *witness* for an obstacle. However, when implemented in the most straightforward way this comparison often leads to severe mapping errors that can cause the vehicle to veer off the road.

The key problem in the analysis of the 3-D point clouds is pose estimation error. In particular, noise in GPS, IMU, and wheel sensors affects the accuracy of the UKF estimates. For a forward-pointed laser measuring at a range of 30 meters, a 1-degree error in pitch induces a 0.5 meter error in perceived Z. As a result, a relatively small UKF error can lead the vehicle to perceive huge fake obstacles where in reality there are none. Such "phantom obstacles" are common when not considering this type of noise. In fact, in one of our reference datasets the number of false positives in the map (phantom obstacles) was found to be 12.6%—despite all our attempts to optimize the UKF. Figure 5 illustrates such a situation.

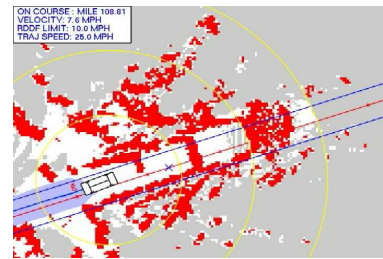The solution to this problem is found in probabilistic anal-

Figure 5: Small errors in pose estimation (smaller than 0.5 degrees) induce massive terrain classification errors, which can force the robot off the road.

ysis. The robot uses a Markov model to model the development of UKF noise over time. The state of this Markov chain is the noise in the vehicle state estimates x, y, and z, and the three Euler angles.

Our approach stipulates that the noise in these variables is itself a Markov chain that slowly changes state (of the noise variables) over time. In particular, at time t there might be a fixed value of the UKF error, where error is defined as the difference between the true state (which cannot be measured) and the UKF state estimate. At time $t + 0.01$sec, the error is essentially the same, plus a zero-mean noise variable with very small variance. Over time, the addition of many such noise variables makes large errors possible. However, in short time intervals, the *change of error* under this Markov chain is small.

The Markov chain makes it possible to evaluate pairs of witnesses probabilistically. The probability that two such points actually correspond to an obstacle is now a function of the time elapsed, since the more time passes by, the more error might have been introduced in the system. The mathematics of such a relative comparison are straightforward for Gaussian noise. The robot accepts obstacles only if two measurements pass the resulting probabilistic test with .95 probability.

While this probabilistic analysis of the data has the potential to reduce the number of false positives, it comes with its own problems. In particular, the Markov chain is characterized by a number of parameters (the drift of the error, the amount of new error introduced as a function of vehicle speed, roll, pitch, shock, and so on). These parameters are the result of complex interactions of sensors, weather, and proprietary software of the sensor manufacturers.

To solve this problem, we employed a discriminative machine learning algorithm. This algorithm "rewarded" the robot for correctly classifying free terrain as free, and occupied terrain as occupied. The training data for this algorithm was collected through human driving. Fig. 6 illustrates the process: here a human driver labels drivable areas by driving over it (colored in blue). A stripe on both sides of the vehicle is assumed to be non-drivable (labeled in red).

The discriminative learning algorithm tunes the parameters of the Markov chain so as to minimize the error in both categories, drivable and non-drivable. The learning algorithm is implemented via coordinate ascent: It jointly optimizes all parameters until a local optimum is reached. The details of the coordinate ascent are beyond the scope of this paper.

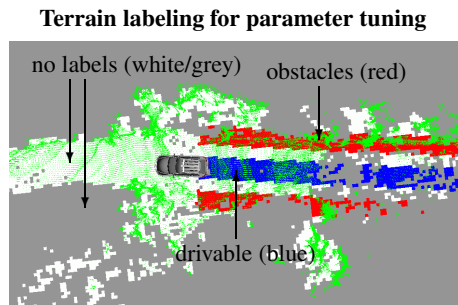In testing, we find that the effect of this probabilistic

Terrain labeling for parameter tuning

Figure 6: Terrain labeling for parameter tuning: The area traversed by the vehicle is labeled as "drivable" (blue) and two stripes at a fixed distance to the left and the right are labeled as "obstacles" (red). While these labels are only approximate, they are extremely easy to obtain and significantly improve the accuracy of the resulting map when used for parameter tuning.

(a) Robot and laser scan plotted over time
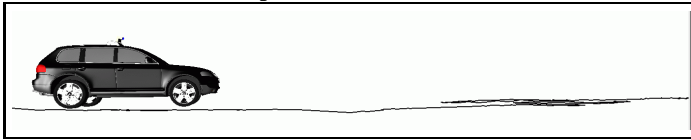
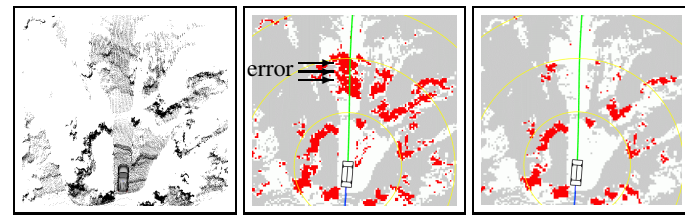**(b) 3-D point cloud**   **(c) straw man method**   **(d) our result**

Figure 7: Comparison of non-probabilistic and probabilistic analysis: (a) shows a scan over time, (b) the 3-D point cloud, (c) the erroneous map and (d) the result of the probabilistic analysis.

method paired with the data-driven tuning algorithm is substantial. In one of the development datasets of hard mountain terrain, the approach reduces the false-positive rate from 12.6% down to 0.02%, while leaving the number of obstacles detected virtually unaffected.

Vision Mapping

Vision is employed to overcome the limited range of laser perception. With a 25 meter range, the laser-based system would only provide safe driving at speeds up to 25 mph. The camera can see much further. However, separating drivable from non-drivable terrain in a camera image is not a trivial task.

To find drivable terrain, our software leverages the laser perception into the vision domain. More specifically, the vision system analyzes the laser map for a drivable region in the near-range. When such a region is found, it projects the drivable area into the field of view of the camera (boxed area in Fig. 8). The vision system then fits a Gaussian that models the color distribution of pixels within the drivable area. It uses this model to identify other areas areas of similar appearance in the image, which are then equally labeled as drivable. In this way, the vision module extends the range of per-
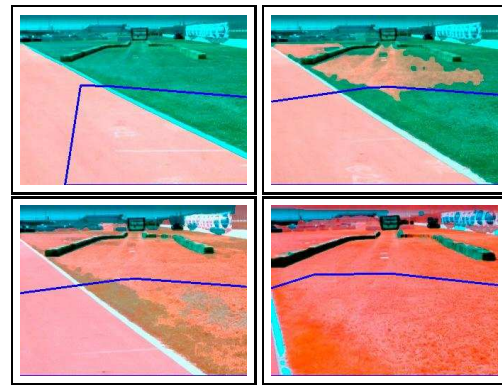

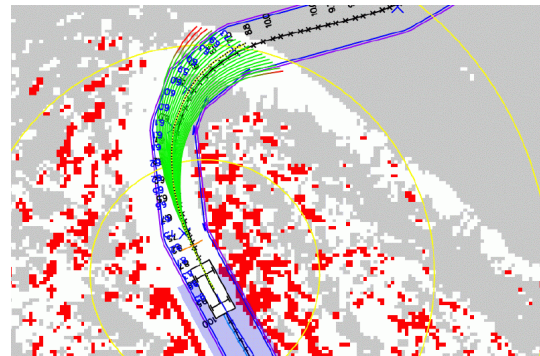Figure 8: Online adaptation to the drivable terrain.

Figure 9: Snapshot of the path planner as it processes the drivability map. This snapshot is taken from the most difficult part of the 2005 DARPA Grand Challenge, a mountainous area called *Beer Bottle Pass*.

ception significantly beyond that of the laser. This approach is reminiscent of prior work in (Dickmanns *et al.* 1994; Pomerleau & Jochem 1996), which also used adaptive techniques for robotic driving. The learning is executed at 8 Hertz, and an exponential filter is used to decay past Gaussian filter parameters over time.

Fig. 8 illustrates the resulting vision system. The data shown here is taken form the *National Qualification Event* of the DARPA Grand Challenge. Here the vehicle moves from a pavement to grass, both of which are drivable. The sequence in Fig. 8 illustrates the adaptation at work: the boxed areas towards the bottom of the image is the training region, and the red coloring in the image is the result of applying the learned classifier. As is easily seen in Fig. 8, the vision module successfully adapts from pavement to to grass within less than a second while still correctly labeling the hay bales and other obstacles.

Since certain color changes are natural even on flat terrain, the vision output is not used for steering control. Instead, it is used exclusively for velocity control. When no drivable corridor is detected within a range of 40 meters, the robot simply slows down to 25 mph, at which point the laser range is sufficient for safe navigation.

Path Planning and Steering Control

The steering of the vehicle is realized by a number of modules in the control layer. Chief among them is the path planning
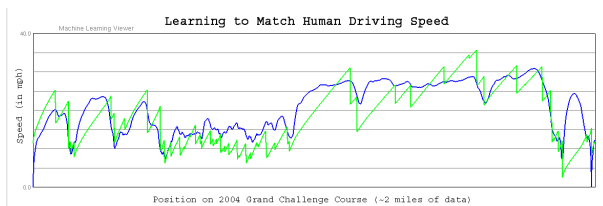
Figure 10: Human velocity compared to the output of the velocity controller, for a 2.2-miles stretch of the 2004 DARPA Grand Challenge course.

module, which determines the desired path of the vehicle. Fig 9 shows an example. The green trajectories are possible paths, generated from a smoothed version of the baseline trajectory, but with changing lateral offsets. By varying the lateral offset, the search for the best path becomes a 1-D search, which can be executed efficiently. To select the final path, the path planner evaluates paths according a number of constraints: the DARPA-imposed corridor, the number of obstacles under a path, and the nearness to the perceived road center.

To accommodate the relatively short range of laser perception, the path planner evaluates two different types of steering actions: *nudges* and *swerves*. Nudges are gradual changes which are often necessary to avoid obstacles on the side of the corridor. Swerves are rapid changes that enables the robot to avoid frontal obstacles.

The best path is continuously communicated to a steering controller, which is responsible for the motor commands to the steering column. The steering controller is a PID controller on the steering wheel direction. The default steering direction is always such that the vehicle front wheels are parallel to the target trajectory. However, lateral error between the desired and actual trajectory results in a change of this default steering direction that is proportional to this error. Additional terms in the PID controller compensate for steering wheel lag, drift, and bias. The planning and steering control modules are executed at 20 Hertz.

Velocity Control

The velocity control module is another key innovation in the software architecture, which we believe to be unique to our vehicle. Intelligent velocity control is essential to avoid fishtailing and to protect the vehicle from the effects of ruts and uneven terrain.

The velocity controller uses a number of hard constraints to set the maximum velocity, which include the DARPA-provided speed limits, and dynamic constraints on maximum lateral accelerations that effectively slow down the vehicle in curves. Additionally, the controller also monitors the terrain and adjusts the velocity dynamically. It does so by monitoring a number of parameters: the terrain slope (pitch/roll), the nearness of obstacles, and the roughness of the terrain.

While slope and obstacles are easy to sense, road roughness is not. At first glance, one might be tempted to use the vehicle accelerometers to measure terrain roughness. However, raw observations from the accelerometers measure not only the high-frequency texture or "roughness" of the road, but also the low frequency forward and lateral accelerations of the vehicle, and the gravity vector. Our approach therefore applies a high-pass filter tuned to accelerations whose mag-

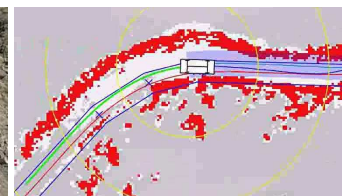**(a) Beer Bottle Pass**   **(b) Map and GPS corridor**

Figure 11: Snapshot of the map acquired by the robot on the "Beer Bottle Pass," the most difficult passage of the DARPA Grand Challenge. The two blue contours mark the GPS corridor provided by DARPA, which aligns poorly with the map data. This analysis suggests that a robot that followed the GPS via points blindly would likely have failed to traverse this narrow mountain pass.
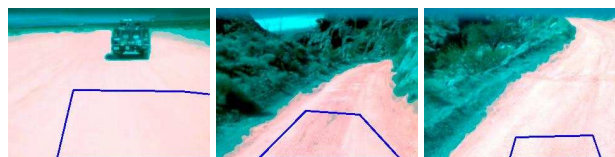

Figure 12: Processed camera images from the race.

nitudes changes at high frequencies. In this way, the filter ignores gravity and accelerations due to vehicle maneuvers. The result of this filter is an estimate of the shocks caused by the road surface conditions.

To control velocity, the vehicle uses an upper bound on the maximum acceptable shock. When an acceleration is observed that violates this bound, the vehicle slows down to a velocity under which the same impact would not have violated this bound. This velocity assumes a linear relationship between vehicle speed and shock. Once the vehicle has slowed down, it gradually increases its speed bound over time.

The resulting velocity controller has two main parameters: the maximum acceptable shock, and the rate at which the velocity bound is lifted over time. Rather than setting those parameters by hand, we used a simple learning algorithm to tune them based on human driving. Figure 10 shows an example of the learned controller. Shown there is the velocity profile of a human driver, compared to the velocity profile of the controller. Both attain approximately equal speeds. It is easy to see that the robot slows down in rough terrain, and gradually recovers so as to return to plausible velocities.

Results from the Race

Before the Grand Challenge, Stanley drove more than 1,000 miles through desert terrain in the southwestern U.S. The longest single stretch of autonomous motion was 200 miles along a large cyclic dirt track, during which the robot encountered approximately 90 frontal obstacles at speeds of up to 35 mph. Before the race, the robot drove 418 miles without any errors that would have required human intervention.

Fig. 14 shows the course and the actual velocities attained by our robot. Stanley finished the DARPA Grand Challenge in 6 hours 53 minutes and 58 seconds, ahead of any other robot in the race. The maximum speed was 38.0 mph and the average speed was 19.1 mph, although early in the race the robot averaged 24.8 mph. The robot was paused twice by the race organizers, for a total of 9 minutes and 20 seconds pause
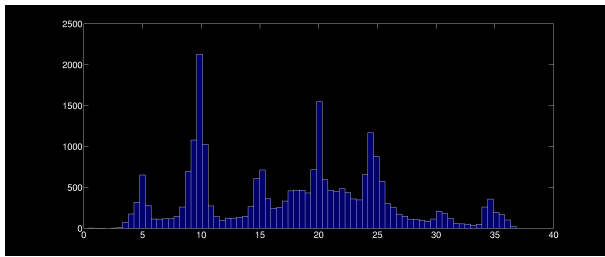
Figure 13: Velocity histogram during the race.

time. 5 hours and 24 minutes into the race, at mile marker 101.5, our robot passed the top-seeded robot in the race and formally took the lead, which it would then maintain until the end of the race.

During 4.7%of the Grand Challenge, the GPS reported 60cm or more of error. During 68.2% of the race course, the velocity was determined as pre-calculated. In the remaining 31.8%, the robot adjusted its velocity because of the terrain roughness (18.1%), the vision-based road finder (13.1%), and GPS blackouts in underpasses (.6%). The final finishing time was of course somewhat unpredictable, and a narrow 11 minutes ahead of the second fastest robot. A histogram of velocities is depicted in Fig. 13.

Unlike most other robots in the race, our robot showed no evidence of collisions during the race. This matched the robot's performance in the National Qualification Event (NQE), where Stanley emerged as the sole contestant that never collided with an obstacle.

A key difference of our approach to some of the other competitors was our robot's ability to vary its speed in response to terrain. For example, some other robots followed a pre-planned velocity profile, in one case developed by dozens of human experts in the hours leading up to the race (Urmson *et al.* 2004). It is difficult to say whether the speed adaptation prevented collisions in the race; however, in our testing we found it to be absolutely essential for perceived safe driving.

Discussion

This paper surveyed the overall software architecture of Stanley, an autonomous robot designed for desert driving. Stanley participated successfully in the DARPA grand Challenge, a challenging offroad race organized by the U.S. government.

The software architecture builds on a decade or more of research in AI. Many of the key software modules relied on adaptive and probabilistic techniques for accommodating sensor noise, and for tuning the parameters. The resulting robot is a competent driver, which has been proven in one of the most challenging races in robotics history.

There are, however, a number of limitations. First and foremost, the robot cannot accommodate moving traffic. This was not a requirement of the DARPA Grand Challenge, but will most certainly be necessary when deploying autonomous driving technology anywhere in the world. Second, the speed of 38 mph, while reasonable for off-road driving, is still relatively low for more urban environments. And third, while the robot managed to avoid collisions, it sometimes reacts later than a human driver would, which results in stronger steering and braking responses. The last two limitations are mostly
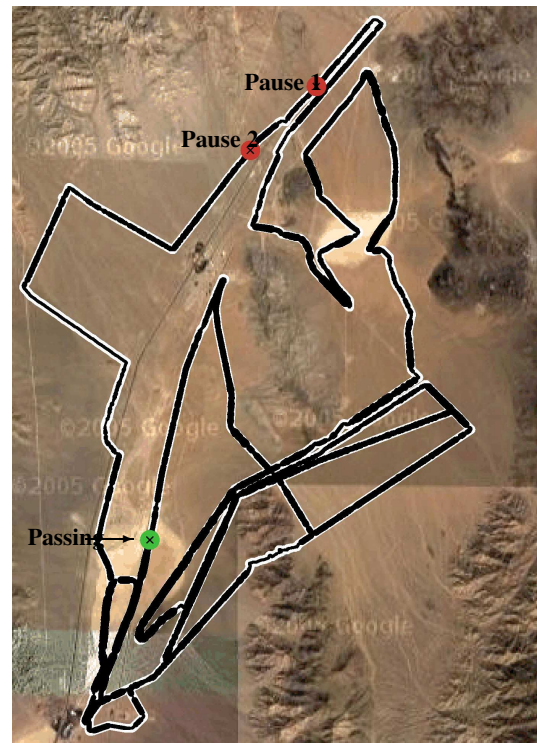

Figure 14: This map shows the DARPA Grand Challenge course. The *thickness* of the trajectory indicates the actual race speed (thicker means faster). At the locations marked by the red 'x's, the race organizers paused our robot because of the close proximity to another robot. At Mile 101.5, this other robot was passed (green 'x').

the result of the relatively short reach of the sensors.

Nevertheless, Stanley successfully finished the 2005 DARPA Grand Challenge, and performed best out of a field of 195 competing teams. We attribute the success of the vehicle largely to its software architecture and the various detailed technical innovations in robot perception and control.

References

Dickmanns, E., et al. The seeing passenger car VaMoRs-P. ISIV-94.

Gat, E. 1998. Three-layered architectures. In Kortenkamp, D.; Bonasso, R.; and Murphy, R., eds., *AI-based Mobile Robots: Case Studies of Successful Robot Systems*. MIT Press.

Grewal, M.; Weill, L.; and Andrews, A. 2001. *Global Positioning Systems, Inertial Navigation, and Integration*. Wiley.

Hebert, M.; Thorpe, C.; and Stentz, A. 1997. *Intelligent Unmanned Ground Vehicles: Autonomous Navigation Research at CCMU*. Kluwer.

Julier, S., and Uhlmann, J. A new extension of the Kalman filter to nonlinear systems. ISADSSC-1997.

Pomerleau, D., and Jochem, T. 1996. Rapidly adapting machine vision for automated vehicle steering. *IEEE Expert* 11(2).

Simmons, R., and Apfelbaum, D. A task description language for robot control. IROS-1998.

Urmson, C., et al. High speed navigation of unrehearsed terrain: Red team technology for grand challenge 2004. TR CMU-RI-TR-04-37.