Stanley: The Robot that Won the DARPA Grand Challenge

Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, and Pascal Stang
Stanford Artificial Intelligence Laboratory
Stanford University
Stanford, California 94305

Sven Strohband, Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe van Niekerk, Eric Jensen, and Philippe Alessandrini
Volkswagen of America, Inc.
Electronics Research Laboratory
4009 Miranda Avenue, Suite 100
Palo Alto, California 94304

Gary Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, and Ara Nefian
Intel Research
2200 Mission College Boulevard
Santa Clara, California 95052

Pamela Mahoney
Mohr Davidow Ventures
3000 Sand Hill Road, Bldg. 3, Suite 290
Menlo Park, California 94025

Received 13 April 2006; accepted 27 June 2006

Journal of Field Robotics 23(9), 661–692 (2006)
Published online in Wiley InterScience (www.interscience.wiley.com). • DOI: 10.1002/rob.20147

This article describes the robot Stanley, which won the 2005 DARPA Grand Challenge. Stanley was developed for high-speed desert driving without manual intervention. The robot's software system relied predominately on state-of-the-art artificial intelligence technologies, such as machine learning and probabilistic reasoning. This paper describes the major components of this architecture, and discusses the results of the Grand Challenge race.

1. INTRODUCTION

The Grand Challenge was launched by the Defense Advanced Research Projects Agency (DARPA) in 2003 to spur innovation in unmanned ground vehicle navigation. The goal of the Challenge was to develop an autonomous robot capable of traversing unrehearsed off-road terrain. The first competition, which carried a prize of $1M, took place on March 13, 2004. It required robots to navigate a 142-mile long course through the Mojave desert in no more than 10 h. 107 teams registered and 15 raced, yet none of the participating robots navigated more than 5% of the entire course. The challenge was repeated on October 8, 2005, with an increased prize of $2M. This time, 195 teams registered and 23 raced. Of those, five teams finished. Stanford's robot "Stanley" finished the course ahead of all other vehicles in 6 h, 53 min, and 58 s, and was declared the winner of the DARPA Grand Challenge; see Figure 1.

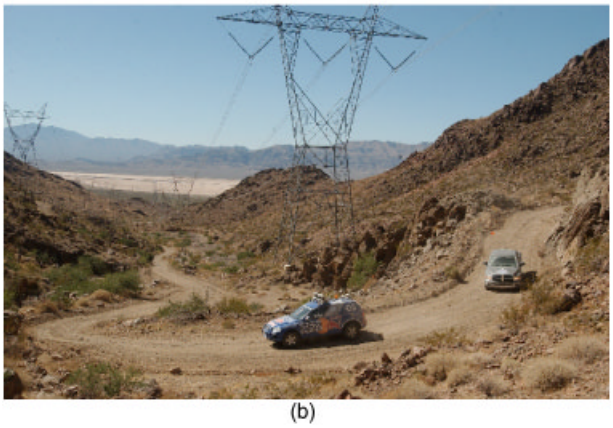
This paper describes the robot Stanley, and its software system in particular. Stanley was developed by a team of researchers to advance the state-of-the-art in autonomous driving. Stanley's success is the result of an intense development effort led by Stanford University, and involving experts from Volkswagen of America, Mohr Davidow Ventures, Intel Research, and a number of other entities. Stanley is based on a 2004 Volkswagen Touareg R5 TDI, outfitted with a six processor computing platform provided by Intel, and a suite of sensors and actuators for autonomous driving. Figure 2 shows images of Stanley during the race.

The main technological challenge in the development of Stanley was to build a highly reliable system, capable of driving at relatively high speeds through diverse and unstructured off-road environments, and to do all this with high precision. These requirements led to a number of advances in the field of autonomous navigation, as surveyed in this paper. Methods were developed, and existing methods extended, in the areas of long-range terrain perception, real-time collision avoidance, and stable vehicle control on slippery and rugged terrain. Many of these developments were driven by the speed requirement, which rendered many classical techniques in the off-road driving field unsuitable. In pursuing these developments, the research team brought to bear algorithms

(a) (b)

Figure 1. (a) At approximately 1:40 pm on Oct 8, 2005, Stanley was the first robot to complete the DARPA Grand Challenge. (b) The robot is being honored by DARPA Director Dr. Tony Tether.

(a)



(b)

Figure 2. Images from the race.

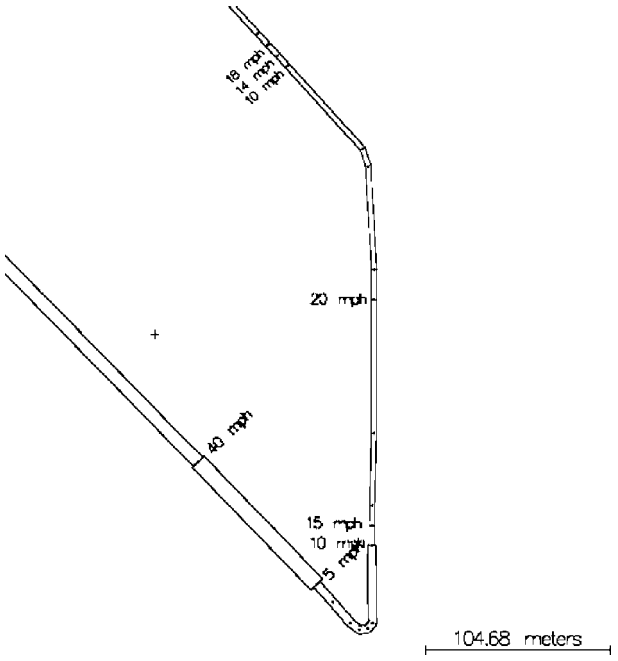


Figure 3. A section of the RDDF file from the 2005 DARPA Grand Challenge. The corridor varies in width and maximum speed. Waypoints are more frequent in turns.

from diverse areas including distributed systems, machine learning, and probabilistic robotics.

1.1. Race Rules

The rules (DARPA, 2004) of the DARPA Grand Challenge were simple. Contestants were required to build autonomous ground vehicles capable of traversing a desert course up to 175-miles long in less than 10 h. The first robot to complete the course in under 10 h would win the challenge and the $2M prize. Absolutely no manual intervention was allowed. The robots were started by DARPA personnel and from that point on had to drive themselves. Teams only saw their robots at the starting line and, with luck, at the finish line.

Both the 2004 and 2005 races were held in the Mojave desert in the southwest United States. The course terrain varied from high-quality graded dirt roads to winding rocky mountain passes; see Figure 2. A small fraction of each course traveled along paved roads. The 2004 course started in Barstow, CA, approximately 100 miles northeast of Los Angeles, and finished in Primm, NV, approximately 30 miles southwest of Las Vegas. The 2005 course both started and finished in Primm, NV.

The specific race course was kept secret from all teams until 2 h before the race. At this time, each team was given a description of the course on CD-ROM in a DARPA-defined route definition data format (RDDF). The RDDF is a list of longitudes, latitudes, and corridor widths that define the course boundary, and a list of associated speed limits; an example segment is shown in Figure 3. Robots that travel substantially beyond the course boundary risk disqualification. In the 2005 race, the RDDF contained 2,935 waypoints.

The width of the race corridor generally tracked the width of the road, varying between 3 and 30 m in the 2005 race. Speed limits were used to protect important infrastructure and ecology along the course, and to maintain the safety of DARPA chase drivers who followed behind each robot. The speed limits varied between 5 and 50 mph. The RDDF defined the approximate route that robots would take,

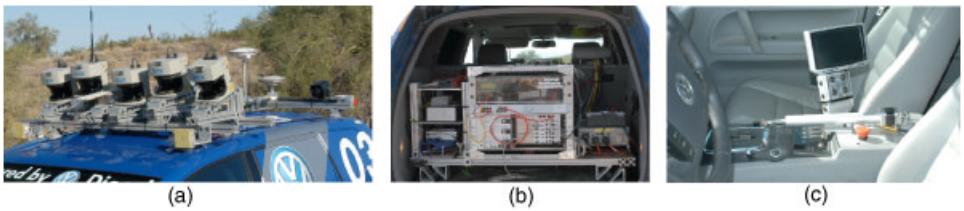


Figure 4. (a) View of the vehicle's roof rack with sensors. (b) The computing system in the trunk of the vehicle. (c) The gear shifter, control screen, and manual override buttons.

so no global path planning was required. As a result, the race was primarily a test of high-speed road finding, obstacle detection, and avoidance in desert terrain.

The robots all competed on the same course; starting one after another at 5 min intervals. When a faster robot overtook a slower one, the slower robot was paused by DARPA officials, allowing the second robot to pass the first as if it were a static obstacle. This eliminated the need for robots to handle the case of dynamic passing.

1.2. Team Composition

The Stanford Racing Team team was organized into four major groups. The *Vehicle Group* oversaw all modifications and component developments related to the core vehicle. This included the drive-by-wire systems, the sensor and computer mounts, and the computer systems. The group was led by researchers from Volkswagen of America's Electronics Research Lab. The *Software Group* developed all software, including the navigation software and the various health monitor and safety systems. The software group was led by researchers affiliated with Stanford University. The *Testing Group* was responsible for testing all system components and the system as a whole, according to a specified testing schedule. The members of this group were separate from any of the other groups. The testing group was led by researchers affiliated with Stanford University. The *Communications Group* managed all media relations and fund raising activities of the Stanford Racing Team. The communications group was led by employees of Mohr Davidow Ventures, with participation from all other sponsors. The operations oversight was provided by a steering board that included all major supporters.

2. VEHICLE

Stanley is based on a diesel-powered Volkswagen Touareg R5. The Touareg has four-wheel drive (4WD), variable-height air suspension, and automatic electronic locking differentials. To protect the vehicle from environmental impact, Stanley has been outfitted with skid plates and a reinforced front bumper. A custom interface enables direct electronic actuation of both the throttle and brakes. A DC motor attached to the steering column provides electronic steering control. A linear actuator attached to the gear shifter shifts the vehicle between drive, reverse, and parking gears [Figure 4(c)]. Vehicle data, such as individual wheel speeds and steering angle, are sensed automatically and communicated to the computer system through a CAN bus interface.

The vehicle's custom-made roof rack is shown in Figure 4(a). It holds nearly all of Stanley's sensors. The roof provides the highest vantage point of the vehicle; from this point, the visibility of the terrain is best, and the access to global positioning system (GPS) signals is least obstructed. For environment perception, the roof rack houses five SICK laser range finders. The lasers are pointed forward along the driving direction of the vehicle, but with slightly different tilt angles. The lasers measure cross sections of the approaching terrain at different ranges out to 25 m in front of the vehicle. The roof rack also holds a color camera for long-range road perception, which is pointed forward and angled slightly downward. For long-range detection of large obstacles, Stanley's roof rack also holds two 24 GHz RADAR sensors, supplied by Smart Microwave Sensors. Both RADAR sensors cover the frontal area up to 200 m, with a coverage angle in azimuth of about 20°. Two antennae of this system are mounted on both sides of the laser sensor array. The lasers, camera, and radar system

comprise the *environment sensor group* of the system. That is, they inform Stanley of the terrain ahead, so that Stanley can decide where to drive, and at what speed.

Further back, the roof rack holds a number of additional antennae: One for Stanley's GPS positioning system and two for the GPS compass. The GPS positioning unit is a L1/L2/Omnistar HP receiver. Together with a trunk-mounted inertial measurement unit (IMU), the GPS systems are the *positioning sensor group*, whose primary function is to estimate the location and velocity of the vehicle relative to an external coordinate system.

Finally, a radio antenna and three additional GPS antennae from the DARPA E-Stop system are also located on the roof. The E-Stop system is a wireless link that allows a chase vehicle following Stanley to safely stop the vehicle in case of emergency. The roof rack also holds a signaling horn, a warning light, and two manual E-stop buttons.

Stanley's computing system is located in the vehicle's trunk, as shown in Figure 4(b). Special air ducts direct air flow from the vehicle's air conditioning system into the trunk for cooling. The trunk features a shock-mounted rack that carries an array of six Pentium M computers, a Gigabit Ethernet switch, and various devices that interface to the physical sensors and the Touareg's actuators. It also features a custom-made power system with backup batteries, and a switch box that enables Stanley to power-cycle individual system components through software. The DARPA-provided E-Stop is located on this rack on additional shock compensation. The trunk assembly also holds the custom interface to the Volkswagen Touareg's actuators: The brake, throttle, gear shifter, and steering controller. A six degree-of-freedom IMU is rigidly attached to the vehicle frame underneath the computing rack in the trunk.

The total power requirement of the added instrumentation is approximately 500 W, which is provided through the Touareg's stock alternator. Stanley's backup battery system supplies an additional buffer to accommodate long idling periods in desert heat.

The operating system run on all computers is Linux. Linux was chosen due to its excellent networking and time sharing capabilities. During the race, Stanley executed the race software on three of the six computers; a fourth was used to log the race data (and two computers were idle). One of the three race computers was entirely dedicated to video processing, whereas the other two executed all other software. The computers were able to poll the sensors at up to 100 Hz, and to control the steering, throttle and brake at frequencies up to 20 Hz.

An important aspect in Stanley's design was to retain street legality, so that a human driver could safely operate the robot as a conventional passenger car. Stanley's custom user interface enables a driver to engage and disengage the computer system at will, even while the vehicle is in motion. As a result, the driver can disable computer control at any time of the development, and regain manual control of the vehicle. To this end, Stanley is equipped with several manual override buttons located near the driver seat. Each of these switches controls one of the three major actuators (brakes, throttle, and steering). An additional central emergency switch disengages all computer control and transforms the robot into a conventional vehicle. While this feature was of no relevance to the actual race (in which no person sat in the car), it proved greatly beneficial during software development. The interface made it possible to operate Stanley autonomously with people inside, as a dedicated safety driver could always catch computer glitches and assume full manual control at any time.

During the actual race, there was of course no driver in the vehicle, and all driving decisions were made by Stanley's computers. Stanley possessed an operational control interface realized through a touch-sensitive screen on the driver's console. This interface allowed Government personnel to shut down and restart the vehicle, if it became necessary.

3. SOFTWARE ARCHITECTURE

3.1. Design Principles

Before both the 2004 and 2005 Grand Challenges, DARPA revealed to the competitors that a stock 4WD pickup truck would be physically capable of traversing the entire course. These announcements suggested that the innovations necessary to successfully complete the challenge would be in designing intelligent driving software, not in designing exotic vehicles. This announcement and the performance of the top finishers in the 2004 race guided the design philosophy of the Stanford Racing Team: *Treat autonomous navigation as a software problem.*

In relation to previous work on robotics architectures, Stanley's software architecture is related to the well-known *three-layer architecture* (Gat, 1998), albeit without a long-term symbolic planning method. A number of guiding principles proved essential in the design of the software architecture:

3.1.1. Control and Data Pipeline

There is no centralized master process in Stanley's software system. All modules are executed at their own pace, without interprocess synchronization mechanisms. Instead, all data are globally time stamped, and time stamps are used when integrating multiple data sources. The approach reduces the risk of deadlocks and undesired processing delays. To maximize the configurability of the system, nearly all interprocess communication is implemented through publish-subscribe mechanisms. The information from sensors to actuators flows in a single direction; no information is received more than once by the same module. At any point in time, all modules in the pipeline are working simultaneously, thereby maximizing the information throughput and minimizing the latency of the software system.

3.1.2. State Management

Even though the software is distributed, the *state* of the system is maintained by local authorities. There are a number of state variables in the system. The health state is locally managed in the health monitor; the parameter state in the parameter server; the global driving mode is maintained in a finite state automaton; and the vehicle state is estimated in the state estimator module. The environment state is broken down into multiple maps (laser, vision, and radar). Each of these maps are maintained in dedicated modules. As a result, all other modules will receive values that are mutually consistent. The exact state variables are discussed in later sections of this paper. All state variables are broadcast to relevant modules of the software system through a publish-subscribe mechanism.

3.1.3. Reliability

The software places strong emphasis on the overall reliability of the robotic system. Special modules monitor the health of individual software and hardware components, and automatically restart or power-cycle such components when a failure is observed. In this way, the software is robust to certain occurrences, such as crashing or hanging of a software modules or stalled sensors.

3.1.4. Development Support

Finally, the software is structured so as to aid development and debugging of the system. The developer can easily run just a subsystem of the software, and effortlessly migrate modules across different processors. To facilitate debugging during the development process, all data are logged. By using a special replay module, the software can be run on recorded data. A number of visualization tools were developed that make it possible to inspect data and internal variables while the vehicle is in motion, or while replaying previously logged data. The development process used a version control process with a strict set of rules for the release of race-quality software. Overall, we found that the flexibility of the software during development was essential in achieving the high level of reliability necessary for long-term autonomous operation.

3.2. Processing Pipeline

The race software consisted of approximately 30 modules executed in parallel (Figure 5). 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 each laser sensor at 75 Hz, from the camera at approximately 12 Hz, the GPS and GPS compass at 10 Hz, and the IMU and the Touareg CAN bus at 100 Hz. This layer also contains a database server with the course coordinates (RDDF file).
2. The **perception layer** maps sensor data into internal models. The primary module in this layer is the unscented Kalman filter (UKF) vehicle state estimator, which determines the vehicle's coordinates, orientation, and velocities. Three different mapping modules build

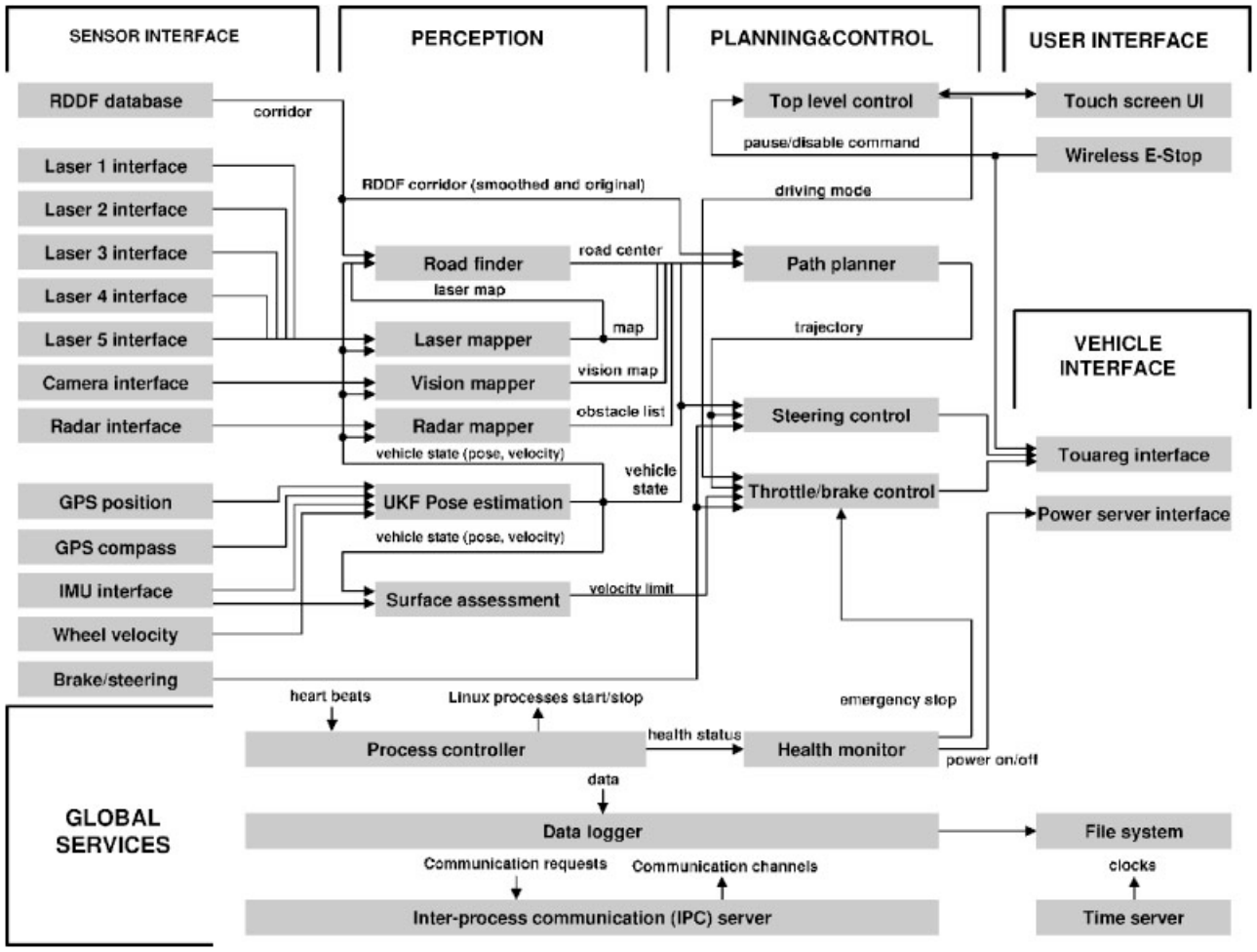


Figure 5. Flowchart of Stanley software system. The software is roughly divided into six main functional groups: Sensor interface, perception, control, vehicle interface, and user interface. There are a number of cross-cutting services, such as the process controller and the logging modules.

two-dimensional (2D) environment maps 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 regulating the steering, throttle, and brake response of the vehicle. A key module is the path planner, which sets the trajectory of the vehicle in steering and velocity space. This trajectory is passed to two closed-loop trajectory tracking controllers, one for the steering control and one for brake and throttle control. Both controllers send low-level commands to the actuators that faithfully execute the trajectory emitted by the planner. 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, and maintains gear state in case backward motion is required.
4. The **vehicle interface layer** serves as the interface to the robot's drive-by-wire system. It

contains all interfaces to the vehicle's brakes, throttle, and steering wheel. It also features the interface to the vehicle's server, a circuit that regulates the physical power to many of the system components.

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 software modules. Naming and communication services are provides through Carnegie Mellon University's (CMU's) interprocess communication toolkit (Simmons & Apfelbaum, 1998). A centralized parameter server maintains a database of all vehicle parameters and updates them in a consistent manner. The physical power of individual system components is regulated by the power server. Another module monitors the health of all systems components and restarts individual system components when necessary. Clock synchronization is achieved through a time server. Finally, a data logging server dumps sensor, control, and diagnostic data to disk for replay and analysis.

The following sections will describe Stanley's core software processes in greater detail. The paper will then conclude with a description of Stanley's performance in the Grand Challenge.

4. VEHICLE STATE ESTIMATION

Estimating vehicle state is a key prerequisite for precision driving. Inaccurate pose estimation can cause the vehicle to drive outside the corridor, or build terrain maps that do not reflect the state of the robot's environment, leading to poor driving decisions. In Stanley, the *vehicle state* comprises a total of 15 variables. The design of this parameter space follows standard methodology (Farrell & Barth, 1999; van der Merwe & Wan, 2004), as indicated in Table I.

Table I. Standard methodology of Stanley's 15 variables.

No. of values	State variable
3	Position (longitude, latitude, and altitude)
3	Velocity
3	Orientation (Euler angles: roll, pitch, and yaw)
3	Accelerometer biases
3	Gyro biases

An unscented Kalman filter (UKF) (Julier & Uhlmann, 1997) estimates these quantities at an update rate of 100 Hz. 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. From a mathematical point of view, the sigma point linearization in the UKF often yields a lower estimation error than the linearization based on Taylor expansion in the extended Kalman filter (EKF) (van der Merwe, 2004). To many, the UKF is also preferable from an implementation standpoint because it does not require the explicit calculation of any Jacobians; although those can be useful for further analysis.

While GPS is available, the UKF uses only a "weak" model. This model corresponds to a moving mass that can move in any direction. Hence, in normal operating mode, the UKF places no constraint on the direction of the velocity vector relative to the vehicle's orientation. Such a model is clearly inaccurate, but the vehicle-ground interactions in slippery desert terrain are generally difficult to model. The moving mass model allows for any slipping or skidding that may occur during off-road driving.

However, this model performs poorly during GPS outages, as the position of the vehicle relies strongly on the accuracy of the IMU's accelerometers. As a consequence, a more restrictive UKF motion model is used during GPS outages. This model constrains the vehicle to only move in the direction it is pointed. The integration of the IMU's gyroscopes for orientation, coupled with wheel velocities for computing the position, is able to maintain accurate pose of the vehicle during GPS outages of up to 2 min long; the accrued error is usually in the order of centimeters. Stanley's health monitor will decrease the maximum vehicle velocity during GPS outages to 10 mph in order to maximize the accuracy of the restricted vehicle model. Figure 6(a) shows the result of position estimation during a GPS outage with the weak vehicle model; Figure 6(b), the result with the strong vehicle model. This experiment illustrates the performance of this filter during a GPS outage. Clearly, accurate vehicle modeling during GPS out-

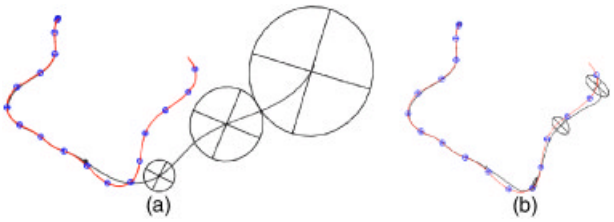


Figure 6. UKF state estimation when GPS becomes unavailable. The area covered by the robot is approximately 100×100 m. The large ellipses illlustrate the position uncertainty after losing GPS. (a) Without integrating the wheel motion the result is highly erroneous. (b) The wheel motion clearly improves the result.

ages is essential. In an experiment on a paved road, we found that even after 1.3 km of travel without GPS on a cyclic course, the accumulated vehicle error was only 1.7 m.

5. LASER TERRAIN MAPPING

5.1. Terrain Labeling

To safely avoid obstacles, Stanley must be capable of accurately detecting nondrivable terrain at a sufficient range to stop or take the appropriate evasive action. The faster the vehicle is moving, the farther away obstacles must be detected. Lasers are used as the basis for Stanley's short and medium range obstacle avoidance. Stanley is equipped with five single-scan laser range finders mounted on the roof, tilted downward to scan the road ahead. Figure 7(a) illustrates the scanning process. Each laser scan generates a vector of 181 range measurements spaced 0.5° apart. Projecting these scans into the global coordinate frame according, to the estimated pose of the vehicle, results in a 3D point cloud for each laser. Figure 7(b) shows an example of the point clouds acquired by the different sensors. The coordinates of such 3D points are denoted $(X_k^i\ Y_k^i\ Z_k^i)$; where k is the time index at which the point was acquired, and i is the index of the laser beam.

Obstacle detection on laser point clouds can be formulated as a classification problem, assigning to each 2D location in a surface grid one of three possible values: Occupied, free, and unknown. A location is occupied by an obstacle if we can find two nearby points whose vertical distance $|Z_k^i - Z_m^j|$ exceeds a critical vertical distance δ. It is considered drivable (free of obstacles) if no such points can be found, but at least one of the readings falls into the corresponding grid cell. If no reading falls into the cell, the drivability of this cell is considered unknown. The search for nearby points is conveniently organized in a 2D grid, the same grid used as the final drivability map that is provided to the vehicle's navigation engine. Figure 8 shows the example grid map. As indicated in this figure, the map assigns terrain to one of three classes: Drivable, occupied, or unknown.

Unfortunately, applying this classification

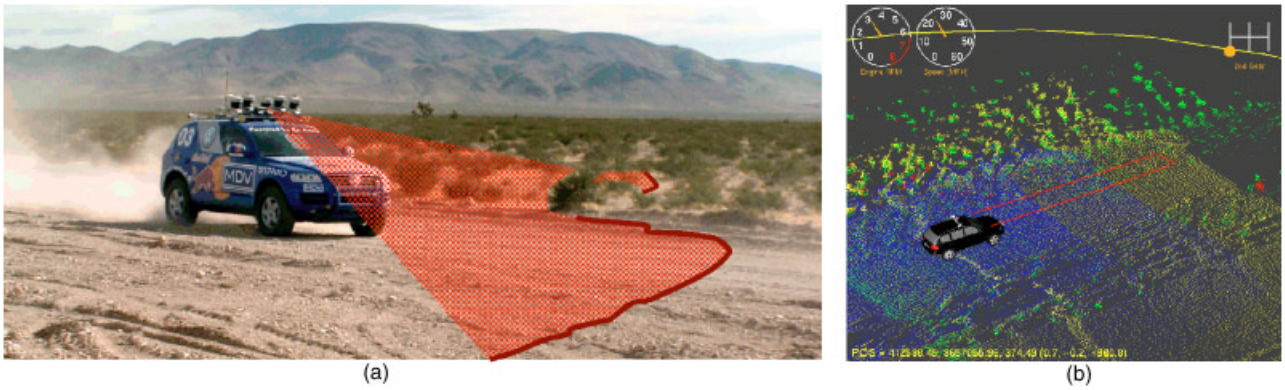


Figure 7. (a) Illustration of a laser sensor: The sensor is angled downward to scan the terrain in front of the vehicle as it moves. Stanley possesses five such sensors, mounted at five different angles. (b) Each laser acquires a three-dimensional (3D) point cloud over time. The point cloud is analyzed for drivable terrain and potential obstacles.

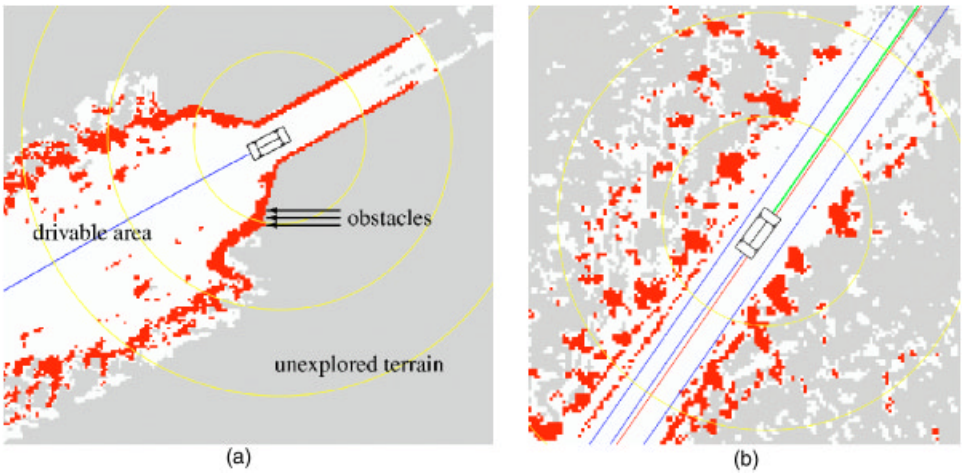


Figure 8. Examples of occupancy maps: (a) An underpass and (b) a road.

scheme directly to the laser data yields results inappropriate for reliable robot navigation. Figure 9 shows such an instance, in which a small error in the vehicle's roll/pitch estimation leads to a massive terrain classification error, forcing the vehicle off the road. Small pose errors are magnified into large errors in the projected positions of laser points because the lasers are aimed at the road up to 30 m in front of the vehicle. In our reference dataset of labeled terrain, we found that 12.6% of known drivable area is classified as obstacle, for a height threshold parameter δ=15 cm. Such situations occur even for roll/pitch errors smaller than 0.5°. Pose errors of this magnitude can be avoided by pose estimation systems that cost hundreds of thousands of dollars, but such a choice was too costly for this project.

The key insight to solving this problem is illustrated in Figure 10. This graph plots the perceived

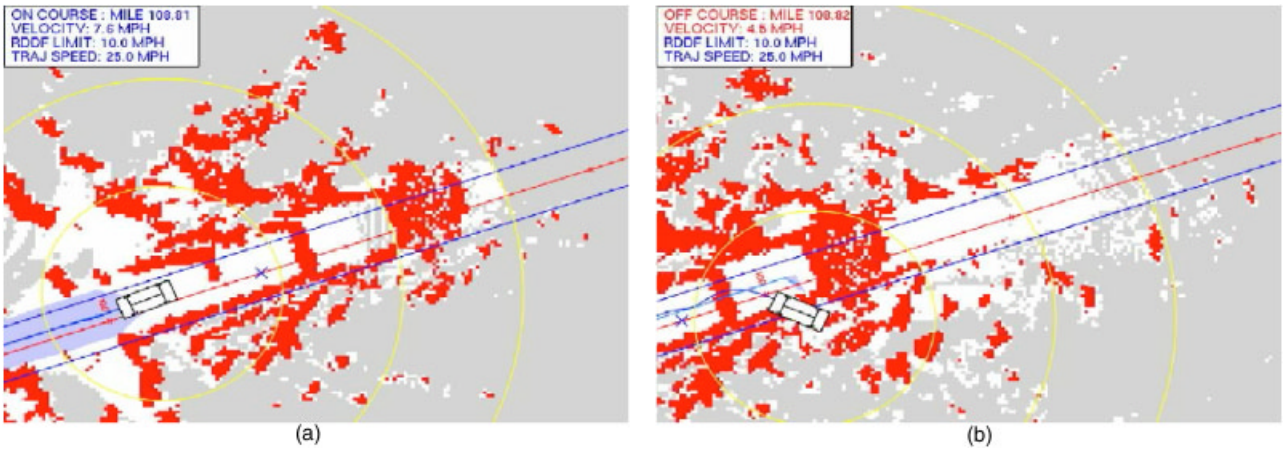


Figure 9. Small errors in pose estimation (smaller than 0.5°) induce massive terrain classification errors, which if ignored could force the robot off the road. These images show two consecutive snapshots of a map that forces Stanley off the road. Here, obstacles are plotted in red, free space in white, and unknown territory in gray. The blue lines mark the corridor as defined by the RDDF.

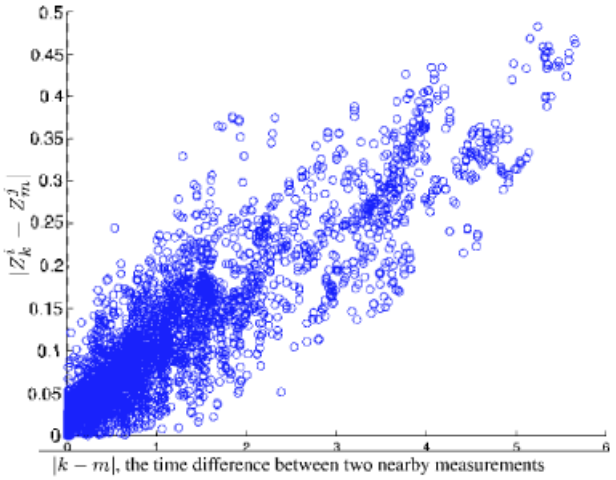


Figure 10. Correlation of time and vertical measurement error in the laser data analysis.

obstacle height $|Z_k^i - Z_m^j|$ along the vertical axis for a collection of grid cells taken from flat terrain. Clearly, for some grid cells, the perceived height is enormous—despite the fact that in reality, the surface is flat. However, this function is not random. The horizontal axis depicts the time difference $\Delta t|k-m|$ between the acquisition of those scans. Obviously, the error is strongly correlated with the elapsed time between the two scans.

To model this error, Stanley uses a first-order Markov model, which models the drift of the pose estimation error over time. The test for the presence of an obstacle is therefore a *probabilistic* test. Given two points $(X_k^i\ Y_k^i\ Z_k^i)^T$ and $(X_m^j\ Y_m^j\ Z_m^j)^T$, the height difference is distributed according to a normal distribution whose variance scales linearly with the time difference $|k-m|$. Thus, Stanley uses a probabilistic test for the presence of an obstacle, of the type

$$p(|Z_k^i - Z_m^j| > \delta) > \alpha, \tag{1}$$

where α is a confidence threshold, e.g., $\alpha = 0.05$.

When applied over a 2D grid, the probabilistic method can be implemented efficiently so that only two measurements have to be stored per grid cell. This is due to the fact that each measurement defines a bound on future Z values for obstacle detection. For example, suppose we observe a measurement for a cell which was previously observed. Then, one or more of three cases will be true:

1. The measurement might be a witness of an obstacle, according to the probabilistic test. In this case, Stanley simply marks the cell as an obstacle and no further testing takes place.
2. The measurement does not trigger as a witness of an obstacle; but, in future tests, it establishes a tighter lower bound on the minimum Z value than the previously stored measurement. In this case, our algorithm simply replaces the previous measurement with this one. The rationale behind this is simple: If the measurement is more restrictive than the previous one, there will not be a situation where a test against this point would fail, while a test against the older one would succeed. Hence, the old point can safely be discarded.
3. The third case is equivalent to the second, but with a refinement of the upper value. A measurement may simultaneously refine the lower and the upper bounds.

The fact that only two measurements per grid cell have to be stored renders this algorithm highly efficient in space and time.

5.2. Data-Driven Parameter Tuning

A final step in developing this mapping algorithm addresses parameter tuning. Our approach and the underlying probabilistic Markov model possess a number of unknown parameters. These parameters include the height threshold δ, the statistical acceptance probability threshold α, and various Markov chain error parameters (the noise covariances of the process noise and the measurement noise).

Stanley uses a discriminative learning algorithm for locally optimizing these parameters. This algorithm tunes the parameters in a way that maximizes the discriminative accuracy of the resulting terrain analysis on labeled training data.

The data are labeled through human driving, similar in spirit to Pomerleau (1993). Figure 11 illustrates the idea: A human driver is instructed to only drive over obstacle-free terrain. Grid cells traversed by the vehicle are then labeled as drivable: This area corresponds to the blue stripe in Figure 11. A stripe to the left and right of this corridor is assumed to be

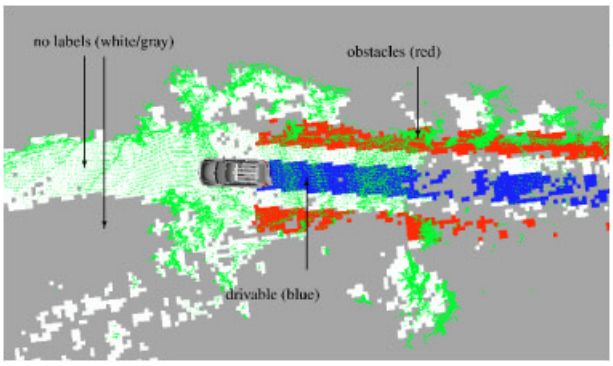


Figure 11. Terrain labeling for parameter tuning: The area traversed by the vehicle is labeled as "drivable" (blue) and two stripes at a fixed distance to the left and the right are labeled as "obstacles" (red). While these labels are only approximate, they are extremely easy to obtain and significantly improve the accuracy of the resulting map when used for parameter tuning.

all obstacles, as indicated by the red stripes in Figure 11. The distance between the drivable and obstacle is set by hand, based on the average road width for a segment of data. Clearly, not all of those cells labeled as obstacles are actually occupied by obstacles; however, even training against an approximate labeling is enough to improve the overall performance of the mapper.

The learning algorithm is now implemented through coordinate ascent. In the outer loop, the algorithm performs coordinate ascent relative to a data-driven scoring function. Given an initial guess, the coordinate ascent algorithm modifies each parameter one after another by a fixed amount. It then determines if the new value constitutes an improvement over the previous value when evaluated over a logged data set, and retains it accordingly. If for a given interval size no improvement can be found, the search interval is cut in half and the search is continued, until the search interval becomes smaller than a preset minimum search interval (at which point the tuning is terminated).

The probabilistic analysis, paired with the discriminative algorithm for parameter tuning, has a significant effect on the accuracy of the terrain labels. Using an independent testing data set, we find that the false positive rate (the area labeled as drivable in Figure 11) drops from 12.6% to 0.002%. At the same time, the rate at which the area off the road is labeled as an obstacle remains approximately constant (from 22.6% to 22.0%). This rate is *not* 100%, simply because most of the terrain there is still flat and drivable. Our approach for data acquisition *mislabels* the flat terrain as nondrivable. Such mislabeling however, does not interfere with the parameter tuning algorithm, and hence is preferable to the tedious process of labeling pixels manually.

Figure 12 shows an example of the mapper in action. A snapshot of the vehicle from the side illustrates that a part of the surface is scanned multiple times due to a change of pitch. As a result, the nonprobabilistic method hallucinates a large occupied area in the center of the road, shown in panel (c) of Figure 12. Our probabilistic approach overcomes this error and generates a map that is good enough for driving. A second example is shown in Figure 13.

6. COMPUTER VISION TERRAIN ANALYSIS

The effective maximum range at which obstacles can be detected with the laser mapper is approximately 22 m. This range is sufficient for Stanley to reliably avoid obstacles at speeds up to 25 mph. Based on the 2004 Race Course, the development team estimated that Stanley would need to reach speeds of 35 mph in order to successfully complete the challenge. To extend the sensor range enough to allow safe driving at 35 mph, Stanley uses a color camera to find drivable surfaces at ranges exceeding that of the laser analysis. Figure 14 compares laser and vision mapping side by side. The left diagram shows a laser map acquired during the race; here, obstacles are detected at an approximate 22 m range. The vision map for the same situation is shown on the right side. This map extends beyond 70 m (each yellow circle corresponds to 10 m range).

Our work builds on a long history of research on road finding (Pomerleau, 1991; Crisman & Thorpe, 1993); see also Dickmanns (2002). To find the road, the vision module classifies images into drivable and nondrivable regions. This classification task is generally difficult, as the road appearance is affected by a number of factors that are not easily measured and change over time, such as the surface material of the road, lighting conditions, dust on the lens of the camera, and so on. This suggests that an adaptive approach is necessary, in which the image interpretation changes as the vehicle moves and conditions change.

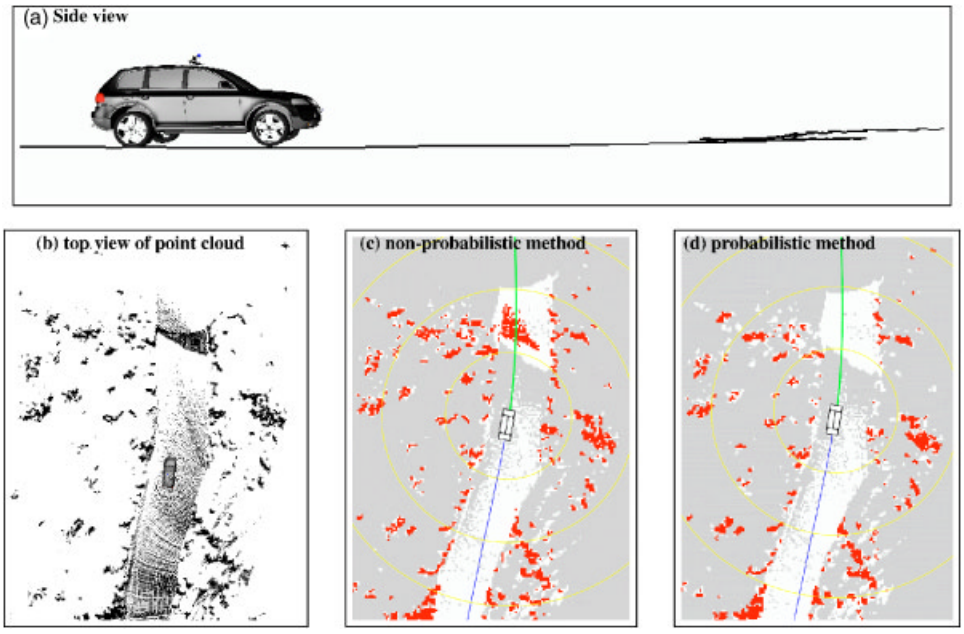


Figure 12. Example of pitching combined with small pose estimation errors: (a) The reading of the center beam of one of the lasers, integrated over time (some of the terrain is scanned twice.); (b) shows 3D point cloud; (c) resulting map without probabilistic analysis, and (d) map with probabilistic analysis. The map shown in (c) possesses a phantom obstacle, large enough to force the vehicle off the road.

The camera images are not the only source of information about upcoming terrain available to the vision mapper. Although we are interested in using vision to classify the drivability of terrain beyond the laser range, we already have such drivability information from the laser in the near range. All that is required from the vision routine is to extend the reach of the laser analysis. This is different from the general-purpose image interpretation problem, in which no such data would be available.

Stanley finds drivable surfaces by projecting drivable area from the laser analysis into the camera

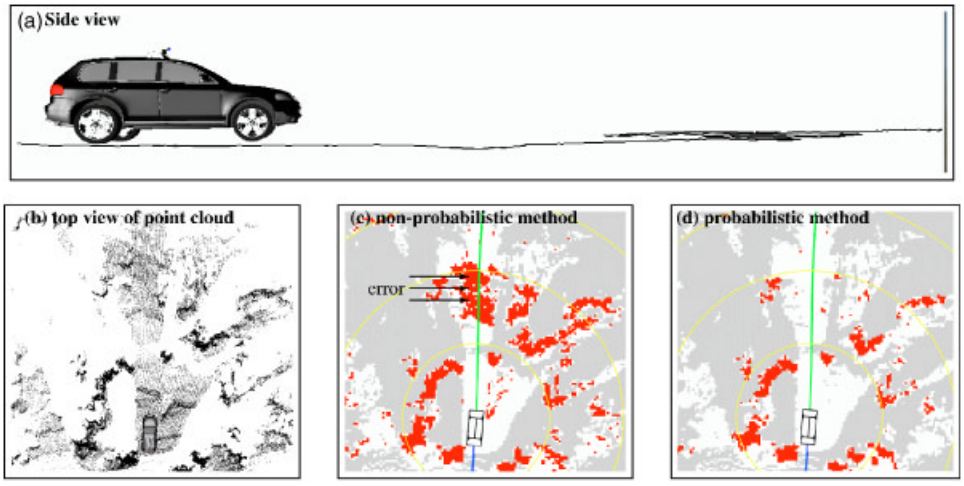


Figure 13. A second example of pitching combined with small page estimation errors.

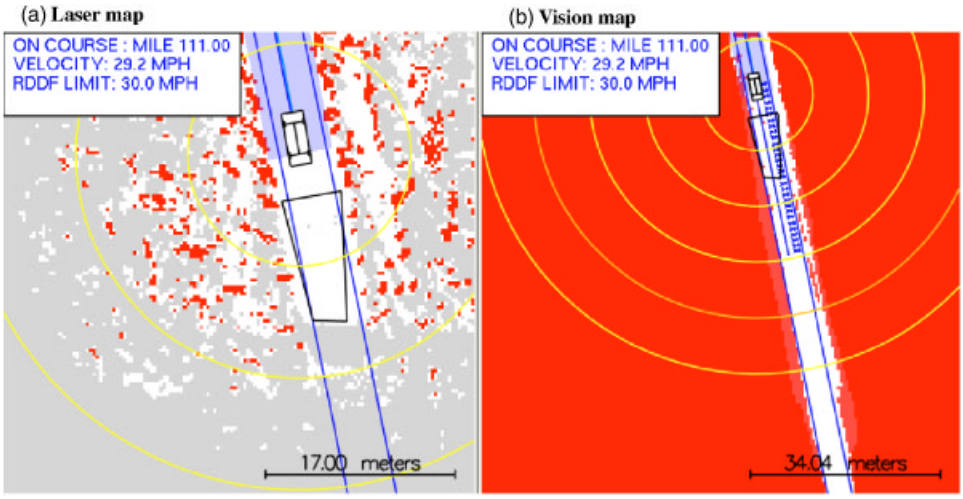


Figure 14. Comparison of the laser-based (left) and the image-based (right) mapper. For scale, circles are spaced around the vehicle at a 10 m distance. This diagram illustrates that the reach of lasers is approximately 22 m, whereas the vision module often looks 70 m ahead.

image. More specifically, Stanley extracts a quadrilateral ahead of the robot in the laser map, so that all grid cells within this quadrilateral are drivable. The range of this quadrilateral is typically between 10 and 20 m ahead of the robot. An example of such a quadrilateral is shown in Figure 14(a). Using straightforward geometric projection, this quadrilateral is then mapped into the camera image, as illustrated in Figures 15(a) and 15(b). An adaptive computer vision algorithm then uses the image pixels inside this quadrilateral as training examples for the concept of drivable surface.

The learning algorithm maintains a mixture of Gaussians that model the color of drivable terrain. Each such mixture is a Gaussian defined in the red/green/blue (RGB) color space of individual pixels; the total number of Gaussians is denoted as n. For each mixture, the learning algorithm maintains a mean RGB color μ_i, a covariance Σ_i, and a count m_i of the total number of image pixels that were used to train this Gaussian.

When a new image is observed, the pixels in the drivable quadrilateral are mapped into a smaller number of k "local" Gaussians using the EM algorithm (Duda & Hart, 1973), with $k < n$ (the covariance of these local Gaussians are inflated by a small value so as to avoid overfitting). These k local Gaussians are then merged into the memory of the learning algorithm, in a way that allows for slow and fast adaptation. The learning adapts to the image in two possible ways; by adjusting the previously found internal Gaussian to the actual image pixels, and by introducing new Gaussians and discarding older ones. Both adaptation steps are essential. The first enables Stanley to adapt to slowly changing lighting

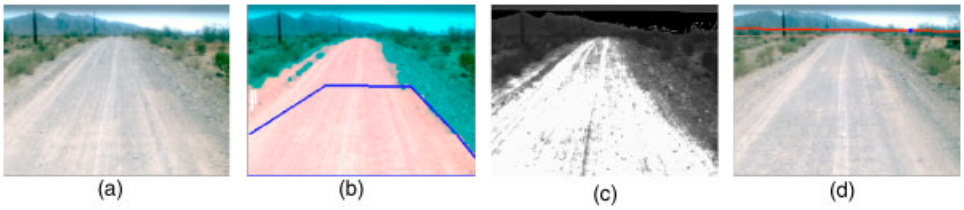


Figure 15. This figure illustrates the processing stages of the computer vision system: (a) a raw image; (b) the processed image with the laser quadrilateral and a pixel classification; (c) the pixel classification before thresholding; and (d) horizon detection for sky removal.

conditions; the second makes it possible to adapt rapidly to a new surface color (e.g., when Stanley moves from a paved to an unpaved road).

In detail, to update the memory, consider the jth local Gaussian. The learning algorithm determines the closest Gaussian in the global memory, where closeness is determined through the Mahalanobis distance

$$d(i,j) = (\mu_i - \mu_j)^T (\Sigma_i + \Sigma_j)^{-1} (\mu_i - \mu_j). \tag{2}$$

Let i be the index of the minimizing Gaussian in the memory. The learning algorithm then chooses one of two possible outcomes:

1. The distance $d(i,j) \leq \phi$, where ϕ is an acceptance threshold. The learning algorithm then assumes that the global Gaussian j is representative of the local Gaussian i, and adaptation proceeds slowly. The parameters of this global Gaussian are set to the weighted mean:

$$\mu_i \leftarrow \frac{m_i \mu_i}{m_i + m_j} + \frac{m_j \mu_j}{m_i + m_j}, \tag{3}$$

$$\Sigma_i \leftarrow \frac{m_i \Sigma_i}{m_i + m_j} + \frac{m_j \Sigma_j}{m_i + m_j}, \tag{4}$$

$$m_i \leftarrow m_i + m_j, \tag{5}$$

   where m_j is the number of pixels in the image that correspond to the jth Gaussian.
2. The distance $d(i,j) > \phi$ for any Gaussian i in the memory. This is the case when none of the Gaussian in memory are near the local Gaussian extracted form the image, where nearness is measured by the Mahalanobis distance. The algorithm then generates a new Gaussian in the global memory, with parameters μ_j, Σ_j, and m_j. If all n slots are already taken in the memory, the algorithm "forgets" the Gaussian with the smallest total pixel count m_i, and replaces it by the new local Gaussian.

After this step, each counter m_i in the memory is discounted by a factor of $\gamma < 1$. This exponential decay term makes sure that the Gaussians in memory can be moved in new directions as the appearance of the drivable surface changes over time.

For finding the drivable surface, the learned Gaussians are used to analyze the image. The image analysis uses an initial sky removal step defined in Ettinger, Nechyba, Ifju & Waszak (2003). A subsequent flood-fill step then removes additional sky pixels not found by the algorithm in Ettinger et al. (2003). The remaining pixels are than classified using the learned mixture of Gaussian, in the straightforward way. Pixels whose RGB value is near one or more of the learned Gaussians are classified as drivable; all other pixels are flagged as nondrivable. Finally, only regions connected to the laser quadrilateral are labeled as drivable.

Figure 15 illustrates the key processing steps. Panel (a) in this figure shows a raw camera image, and panel (b) shows the image after processing. Pixels classified as drivable are colored red, whereas nondrivable pixels are colored blue. The remaining two panels in Figure 15 show intermediate processing steps: The classification response before thresholding [panel (c)] and the result of the sky finder [panel (d)].

Due to the ability to create new Gaussians on the fly, Stanley's vision routine can adapt to new terrain within seconds. Figure 16 shows data acquired at the *National Qualification Event* (NQE) of the DARPA Grand Challenge. Here the vehicle moves from the pavement to grass, both of which are drivable. The sequence in Figure 16 illustrates the adaptation at work: The boxed areas toward the bottom of the image are the training region, and the red coloring in the image is the result of applying the learned classifier. As is easily seen in Figure 16, the vision module successfully adapts from pavement to grass within less than 1 s, while still correctly labeling the hay bales and other obstacles.

Under slowly changing lighting conditions, the system adapts more slowly to the road surface, making extensive use of past images in classification. This is illustrated in the bottom row of Figure 17, which shows results for a sequence of images acquired at the Beer Bottle pass, the most difficult passage in the 2005 Race. Here, most of the terrain has a similar visual appearance. The vision module, however, still competently segments the road. Such a result is only possible because the system balances the use of past images with its ability to adapt to new camera images.

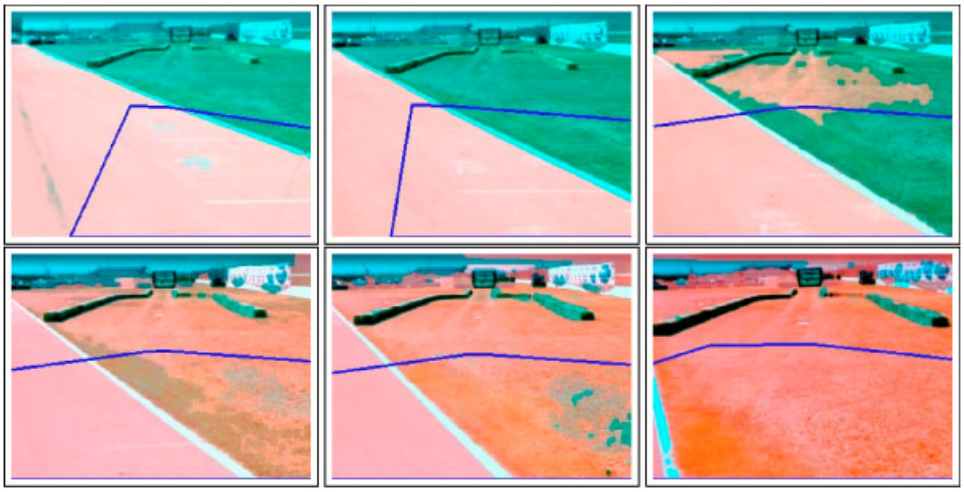


Figure 16. These images illustrate the rapid adaptation of Stanley's computer vision routines. When the laser predominately screens the paved surface, the grass is not classified as drivable. As Stanley moves into the grass area, the classification changes. This sequence of images also illustrates why the vision result should not be used for steering decisions, in that the grass area is clearly drivable, yet Stanley is unable to detect this from a distance.

Once a camera image has been classified, it is mapped into an overhead map, similar to the 2D map generated by the laser. We already encountered such a map in Figure 14(b), which depicted the map of a straight road. Since certain color changes are natural even on flat terrain, the vision map 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 m, the robot simply slows down to 25 mph, at which point the laser range is sufficient for safe navigation. In other words, the vision analysis serves as an early warning system for obstacles beyond the range of the laser sensors.

In developing the vision routines, the research team investigated a number of different learning algorithms. One of the primary alternatives to the generative mixture of the Gaussian method was a discriminative method, which uses boosting and decision stumps for classification (Davies & Lienhart, 2006). This method relies on examples of nondrivable

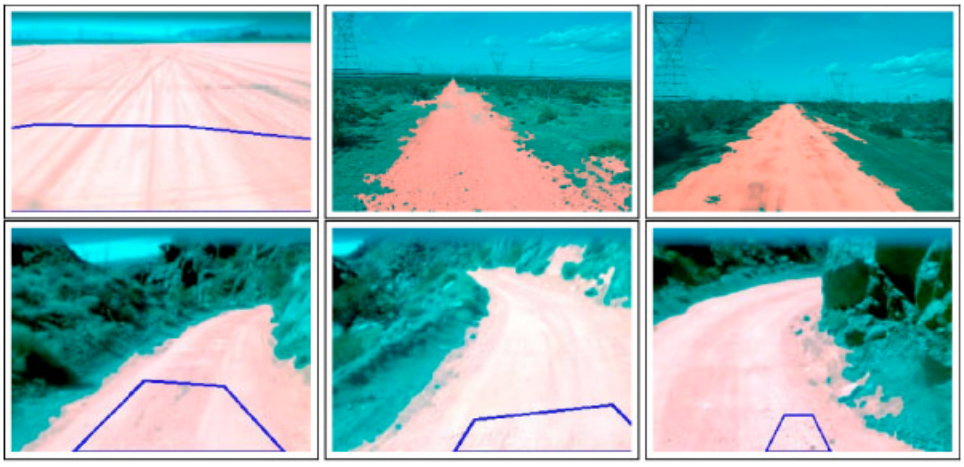


Figure 17. Processed camera images in flat and mountainous terrain (Beer Bottle Pass).

Table II. Road detection rate for the two primary machine learning methods, broken down into different ranges. The comparison yields no conclusive winner.

	Flat desert roads		Mountain roads	
Drivable terrain detection rate (m)	Discriminative training (%)	Generative training (%)	Discriminative training (%)	Generative training (%)
10–20	93.25	90.46	80.43	88.32
20–35	95.90	91.18	76.76	86.65
35–50	94.63	87.97	70.83	80.11
50+	87.13	69.42	52.68	54.89
False positives, all ranges	3.44	3.70	0.50	2.60

terrain, which were extracted using an algorithm similar to the one for finding a drivable quadrilateral. A performance evaluation, carried out using independent test data gathered on the 2004 Race Course, led to inconclusive results. Table II shows the classification accuracy for both methods; for flat desert roads and mountain roads. The generative mixture of Gaussian methods was finally chosen because it does not require training examples of nondrivable terrain, which can be difficult to obtain in flat open lakebeds.

7. ROAD PROPERTY ESTIMATION

7.1. Road Boundary

One way to avoid obstacles is to detect them and drive around them. This is the primary function of the laser mapper. Another effective method is to drive in such a way that minimizes the a priori chances of encountering an obstacle. This is possible because obstacles are rarely uniformly distributed in the world. On desert roads, obstacles–such as rocks, brush, and fence posts–exist most often along the sides of the road. By simply driving down the middle of the road, most obstacles on desert roads can be avoided without ever detecting them!

One of the most beneficial components of Stanley's navigation routines, thus, is a method for staying near the center of the road. To find the road center, Stanley uses probabilistic low-pass filters to determine both road sides based using the laser map. The idea is simple; In expectation, the road sides are parallel to the RDDF. However, the exact lateral offset of the road boundary to the RDDF center is unknown and varies over time. Stanley's low-pass filters are implemented as one-dimensional Kalman filters (KFs). The state of each filter is the lateral distance between the road boundary and the center of the RDDF. The KFs search for possible obstacles along a discrete search pattern orthogonal to the RDDF, as shown in Figure 18(a). The largest free offset is the "observation" to the KF, in that it establishes the local measurement of the road boundary. So, if multiple parallel roads exist in Stanley's field of view, separated by a small berm, the filter will only trace the innermost drivable area.

By virtue of KF integration, the road boundaries change slowly. As a result, small obstacles—or momentary situations without side obstacles—affect the road boundary estimation only minimally; however, persistent obstacles that occur over an extended period of time do have a strong effect.

Based on the output of these filters, Stanley defines the road to be the center of the two boundaries. The road center's lateral offset is a component in scoring trajectories during path planning, as will be discussed further below. In the absence of other contingencies, Stanley slowly converges to the estimated road center. Empirically, we found that this driving technique stays clear of the vast majority of natural obstacles on desert roads. While road centering is clearly only a heuristic, we found it to be highly effective in extensive desert tests.

Figure 18(b) shows an example result of the road estimator. The blue corridor shown there is Stanley's best estimate of the road. Notice that the corridor is confined by two small berms, which are both detected by the laser mapper. This module plays an important role in Stanley's ability to negotiate desert roads.

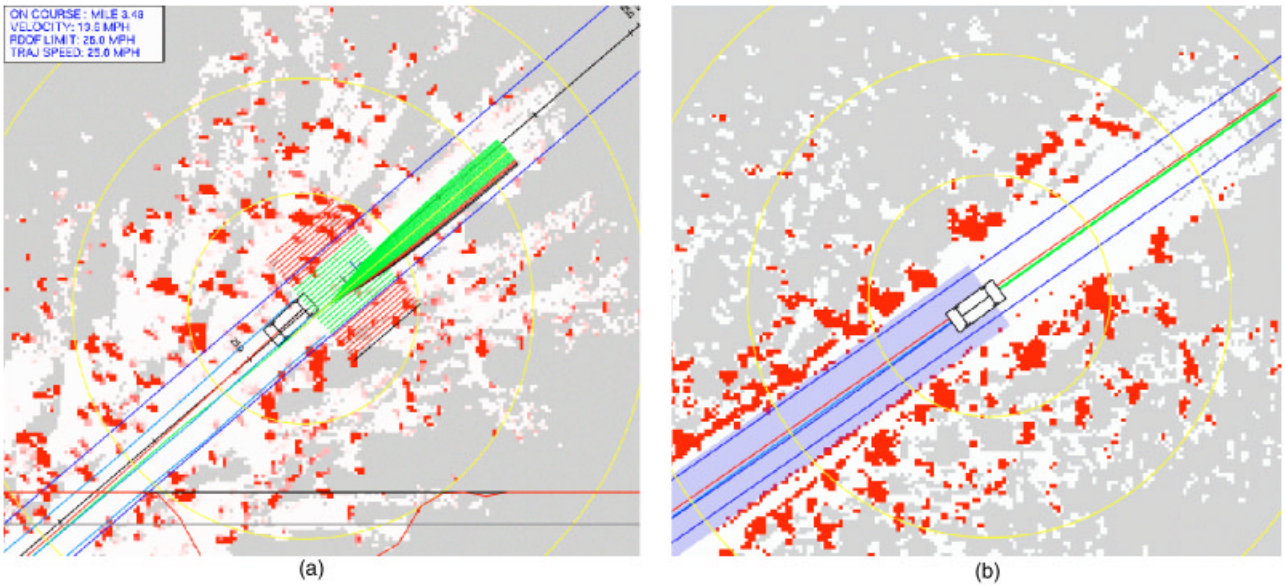


Figure 18. (a) Search regions for the road detection module: The occurrence of obstacles is determined along a sequence of lines parallel to the RDDF; and (b) the result of the road estimator is shown in blue, behind the vehicle. Notice that the road is bounded by two small berms.

7.2. Terrain Ruggedness

In addition to avoiding obstacles and staying centered along the road, another important component of safe driving is choosing an appropriate velocity (Iagnemma & Dubowsky, 2004). Intuitively speaking, desert terrain varies from flat and smooth to steep and rugged. The type of the terrain plays an important role in determining the maximum safe velocity of the vehicle. On steep terrain, driving too fast may lead to fishtailing or sliding. On rugged terrain, excessive speeds may lead to extreme shocks that can damage or destroy the robot. Thus, sensing the terrain type is essential for the safety of the vehicle. In order to address these two situations, Stanley's velocity controller constantly estimates terrain slope and ruggedness and uses these values to set intelligent maximum speeds.

The terrain slope is taken directly from the vehicle's pitch estimate, as computed by the UKF. Borrowing from Brooks & Iagnemma (2005), the terrain ruggedness is measured using the vehicle's z accelerometer. The vertical acceleration is band-pass filtered to remove the effect of gravity and vehicle vibration, while leaving the oscillations in the range of the vehicle's resonant frequency. The amplitude of the resulting signal is a measurement of the vertical shock experienced by the vehicle due to excitation by the terrain. Empirically, this filtered acceleration appears to vary linearly with velocity. (see Figure 19). In other words, doubling the maximum speed of the vehicle over a section of terrain will approximately double the maximum differential acceleration imparted on the vehicle. In Sec. 9.1, this relationship will be used to derive a simple rule for setting maximum velocity to approximately bound the maximum shock imparted on the vehicle.

8. PATH PLANNING

As was previously noted, the RDDF file provided by DARPA largely eliminates the need for any global path planning. Thus, the role of Stanley's path planner is primarily local obstacle avoidance. Instead of planning in the global coordinate frame, Stanley's path planner was formulated in a unique coordinate system: Perpendicular distance, or "lateral offset" to a fixed base trajectory. Varying the lateral offset moves Stanley left and right with respect to the base trajectory, much like a car changes lanes on a highway. By intelligently changing the lateral offset, Stanley can avoid obstacles at high speeds while making fast progress along the course.

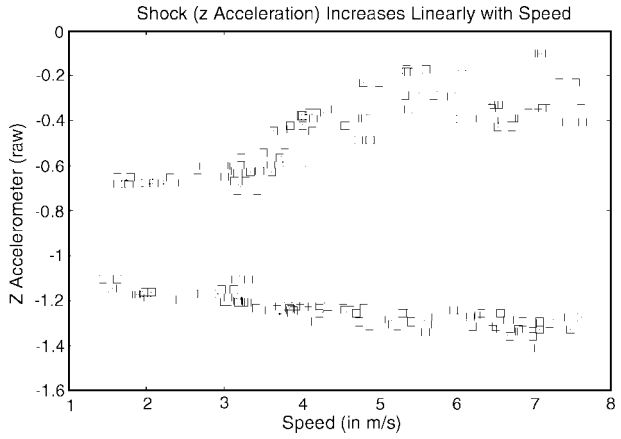


Figure 19. The relationship between velocity and imparted acceleration from driving over a fixed-sized obstacle at varying speeds. The plot shows two distinct reactions to the obstacle; one up and one down. While this relation is ultimately nonlinear, it is well modeled by a linear function within the range relevant for desert driving.

The base trajectory that defines lateral offset is simply a smoothed version of the skeleton of the RDDF corridor. It is important to note that this base trajectory is not meant to be an optimal trajectory in any sense; it serves as a baseline coordinate system upon which obstacle avoidance maneuvers are continuously layered. The following two sections will describe the two parts to Stanley's path planning software: The path smoother that generates the base trajectory before the race, and the online path planner which is constantly adjusting Stanley's trajectory.

8.1. Path Smoothing

Any path can be used as a base trajectory for planning in lateral offset space. However, certain qualities of base trajectories will improve overall performance.

- **Smoothness.** The RDDF is a coarse description of the race corridor and contains many sharp turns. Blindly trying to follow the RDDF waypoints would result in both significant overshoot and high lateral accelerations, both of which could adversely affect vehicle safety. Using a base trajectory that is smoother than the original RDDF will allow Stanley to travel faster in turns and follow the intended course with higher accuracy.
- **Matched curvature.** While the RDDF corridor is parallel to the road in expectation, the curvature of the road is poorly predicted by the RDDF file in turns, again due to the finite number of waypoints. By default, Stanley will prefer to drive parallel to the base trajectory, so picking a trajectory that exhibits curvature that better matches the curvature of the underlying desert roads will result in fewer changes in lateral offset. This will also result in smoother, faster driving.

Stanley's base trajectory is computed before the race in a four-stage procedure.

1. First, points are added to the RDDF in proportion to the local curvature [see Figure 20(a)].
2. The coordinates of all points in the upsampled trajectory are then adjusted through least-squares optimization. Intuitively, this optimization adjusts each waypoint, so as to minimize the curvature of the path while staying as close as possible to the waypoints in the original RDDF. The resulting trajectory is still piece-wise linear, but it is significantly smoother than the original RDDF.
   Let $x_1, \ldots, x_N$ be the waypoints of the base trajectory to be optimized. For each of these points, we are given a corresponding point along the original RDDF, which shall be denoted y_i. The points $x_1, \ldots, x_N$ are obtained by minimizing the following additive function:

$$\begin{aligned} &\underset{x_1,\ldots,x_N}{\operatorname{argmin}} \sum_i |y_i - x_i|^2 \\ &\quad - \beta \sum_n \frac{(x_{n+1} - x_n) \cdot (x_n - x_{n-1})}{|x_{n+1} - x_n||x_n - x_{n-1}|} \\ &\quad + \sum_n f_{\text{RDDF}}(x_n), \end{aligned} \tag{6}$$

   where $|y_i - x_i|^2$ is the quadratic distance between the waypoint x_i and the corresponding RDDF anchor point y_i, and the index variable i iterates over the set of points x_i. Minimizing

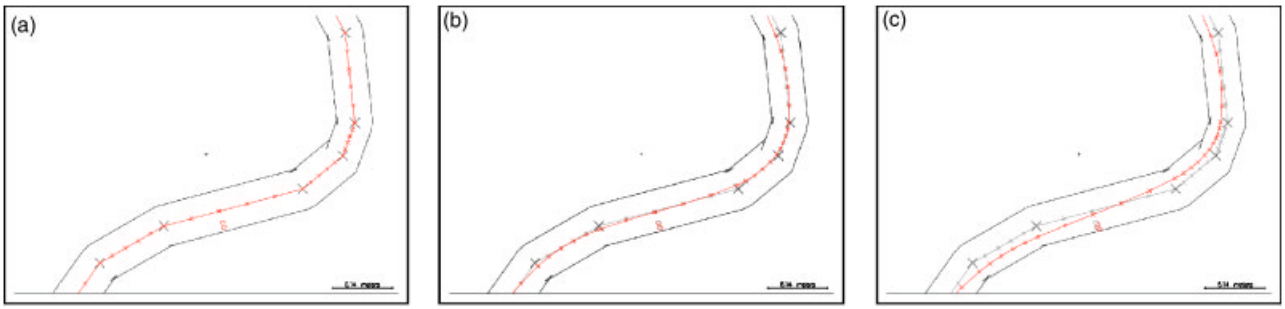


Figure 20. Smoothing of the RDDF: (a) Adding additional points; (b) the trajectory after smoothing (shown in red); (c) a smoothed trajectory with a more aggressive smoothing parameter. The smoothing process takes only 20 seconds for the entire 2005 course.

this quadratic distance for all points i ensures that the base trajectory stays close to the original RDDF. The second expression in Eq. (6) is a curvature term; It minimizes the angle between two consecutive line segments in the base trajectory by minimizing the dot product of the segment vectors. Its function is to smooth the trajectory: The smaller the angle, the smoother the trajectory. The scalar β trades off these two objectives, and is a parameter in Stanley's software. The function $f_{\text{RDDF}}(x_n)$ is a differentiable barrier function that goes to infinity as a point x_n approaches the RDDF boundary, but is near zero inside the corridor away from the boundary. As a result, the smoothed trajectory is always inside the valid RDDF corridor. The optimization is performed with a fast version of conjugate gradient descent, which moves RDDF points freely in 2D space.

3. The next step of the path smoother involves cubic spline interpolation. The purpose of this step is to obtain a path that is differentiable. This path can then be resampled efficiently.
4. The final step of path smoothing pertains to the calculation of the speed limit attached to each waypoint of the smooth trajectory. Speed limits are the minimum of three quantities: (1) The speed limit from corresponding segment of the original RDDF, (2) a speed limit that arises from a bound on lateral acceleration, and (3) a speed limit that arises from a bounded deceleration constraint. The lateral acceleration constraint forces the vehicle to slow down appropriately in turns. When computing these limits, we bound the lateral acceleration of the vehicle to $0.75\ \text{m/s}^2$, in order to give the vehicle enough maneuverability to safely avoid obstacles in curved segments of the course. The bounded deceleration constraint forces the vehicle to slow down in anticipation of turns and changes in DARPA speed limits.

Figure 20 illustrates the effect of smoothing on a short segment of the RDDF. Panel (a) shows the RDDF and the upsampled base trajectory before smoothing. Panels (b) and (c) show the trajectory after smoothing (in red), for different values of the parameter β. The entire data preprocessing step is fully automated, and requires only approximately 20 s of computation time on a 1.4 GHz laptop, for the entire 2005 Race Course. This base trajectory is transferred onto Stanley, and the software is ready to go. No further information about the environment or the race is provided to the robot.

It is important to note that Stanley does not modify the original RDDF file. The base trajectory is only used as the coordinate system for obstacle avoidance. When evaluating whether particular trajectories stay within the designated race course, Stanley checks against the original RDDF file. In this way, the preprocessing step does not affect the interpretation of the corridor constraint imposed by the rules of the race.

8.2. Online Path Planning

Stanley's online planning and control system is similar to the one described in Kelly & Stentz (1998). The online component of the path planner is responsible for determining the actual trajectory of the vehicle during the race. The goal of the planner is to complete the course as fast as possible, while successfully avoiding obstacles and staying inside the RDDF corridor. In the absence of obstacles, the planner will maintain a constant lateral offset from the base trajectory. This results in driving a path parallel to the base trajectory, but possibly shifted left or right. If an obstacle is encountered, Stanley will plan a smooth change in lateral offset that avoids the obstacle and can be safely executed. Planning in lateral offset space also has the advantage that it gracefully handles GPS error. GPS error may systematically shift Stanley's position estimate. The path planner will simply adjust the lateral offset of the current trajectory to recenter the robot in the road.

The path planner is implemented as a search algorithm that minimizes a linear combination of continuous cost functions, subject to a fixed vehicle model. The vehicle model includes several kinematic and dynamic constraints including maximum lateral acceleration (to prevent fishtailing), maximum steering angle (a joint limit), maximum steering rate (maximum speed of the steering motor), and maximum deceleration. The cost functions penalize running over obstacles, leaving the RDDF corridor, and the lateral offset from the current trajectory to the sensed center of the road surface. The soft constraints induce a ranking of admissible trajectories. Stanley chooses the best such trajectory. In calculating the total path costs, unknown territory is treated the same as drivable surface, so that the vehicle does not swerve around unmapped spots on the road, or specular surfaces, such as puddles.

At every time step, the planner considers trajectories drawn from a 2D space of maneuvers. The first dimension describes the amount of lateral offset to be added to the current trajectory. This parameter allows Stanley to move left and right, while still staying essentially parallel to the base trajectory. The second dimension describes the rate at which Stanley will attempt to change to this lateral offset. The lookahead distance is speed dependent and ranges from 15 to 25 m. All candidate paths are run through the vehicle model to ensure that obey the kinematic and dynamic vehicle constraints. Repeatedly layering these simple maneuvers on top of the base trajectory can result in quite sophisticated trajectories.

The second parameter in the path search allows the planner to control the urgency of obstacle avoidance. Discrete obstacles in the road, such as rocks or fence posts, often require the fastest possible change in lateral offset. Paths that change lateral offset as fast as possible without violating the lateral acceleration constraint are called swerves. Slow changes in the positions of road boundaries require slow smooth adjustment to the lateral offset. Trajectories with the slowest possible change in lateral offset for a given planning horizon are called nudges. Swerves and nudges span a spectrum of maneuvers appropriate for high-speed obstacle avoidance: Fast changes for avoiding head on obstacles, and slow changes for smoothly tracking the road center. Swerves and nudges are illustrated in Figure 21. On a straight road, the resulting trajectories are similar to those of Ko & Simmons's (1998) lane curvature method.

The path planner is executed at 10 Hz. The path planner is ignorant to actual deviations from the vehicle and the desired path, since those are handled by the low-level steering controller. The resulting trajectory is therefore always continuous. Fast changes in lateral offset (swerves) will also include braking in order to increase the amount of steering the vehicle can do without violating the maximum lateral acceleration constraint.

Figure 22 shows an example situation for the path planner. Shown here is a situation taken from Beer Bottle Pass, the most difficult passage of the 2005 Grand Challenge. This image only illustrates one of the two search parameters: The lateral offset. It illustrates the process through which trajectories are generated by gradually changing the lateral offset relative to the base trajectory. By using the base trajectory as a reference, path planning can take place in a low-dimensional space, which we found to be necessary for real-time performance.

9. REAL-TIME CONTROL

Once the intended path of the vehicle has been determined by the path planner, the appropriate throttle, brake, and steering commands necessary to

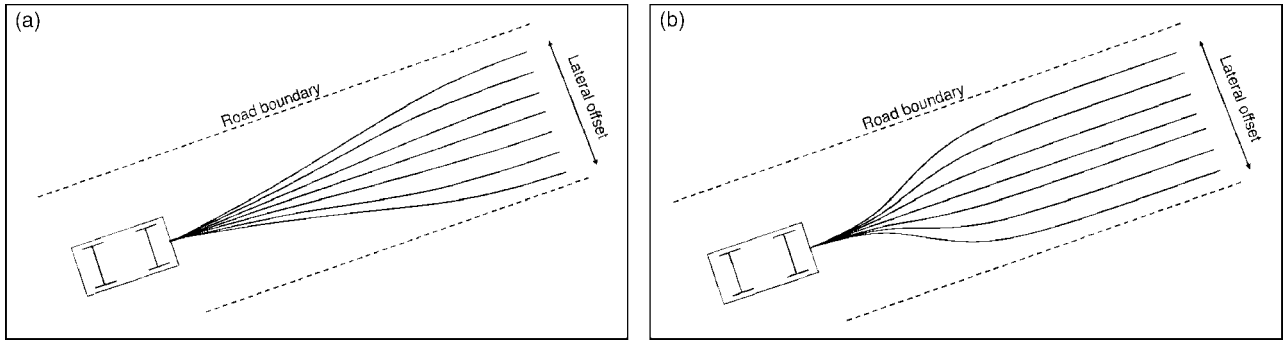


Figure 21. Path planning in a 2D search space: (a) Paths that change lateral offsets with the minimum possible lateral acceleration (for a fixed plan horizon); and (b) the same for the maximum lateral acceleration. The former are called "nudges," and the latter are called "swerves."

achieve that path must be computed. This control problem will be described in two parts: The velocity controller and steering controller.

9.1. Velocity Control

Multiple software modules have input into Stanley's velocity, most notably the path planner, the health monitor, the velocity recommender, and the low-level velocity controller. The low-level velocity controller translates velocity commands from the first three modules into actual throttle and brake commands. The implemented velocity is always the minimum of the three recommended speeds. The path planner will set a vehicle velocity based on the base trajectory speed limits and any braking due to

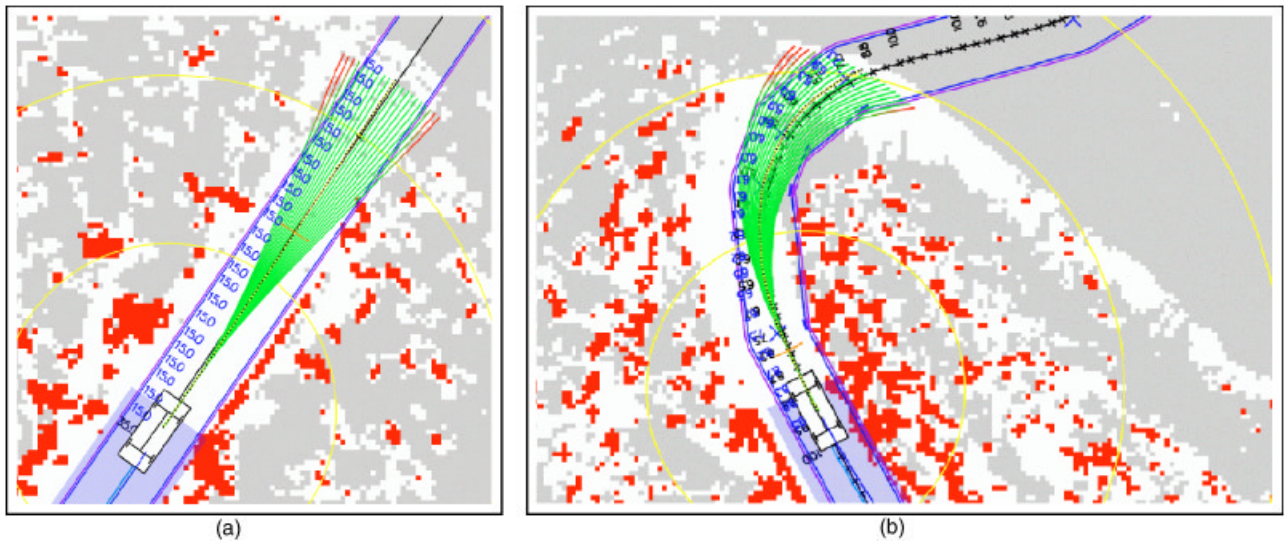


Figure 22. Snapshots of the path planner as it processes the drivability map. Both snapshots show a map, the vehicle, and the various nudges considered by the planner. The first snapshot stems from a straight road (Mile 39.2 of the 2005 Race Course). Stanley is traveling 31.4 mph; and hence, can only slowly change lateral offsets due to the lateral acceleration constraint. The second example is taken from the most difficult part of the 2005 DARPA Grand Challenge, a mountainous area called Beer Bottle Pass. Both images show only nudges for clarity.

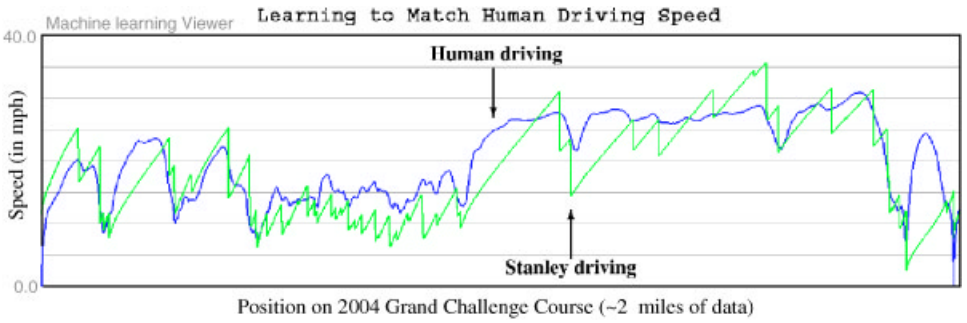


Figure 23. Velocity profile of a human driver and of Stanley's velocity controller in rugged terrain. Stanley identifies controller parameters that match human driving. This plot compares human driving with Stanley's control output.

swerves. The vehicle health monitor will lower the maximum velocity due to certain preprogrammed conditions, such as GPS blackouts or critical system failures.

The velocity recommender module sets an appropriate maximum velocity based on estimated terrain slope and roughness. The terrain slope affects the maximum velocity if the pitch of the vehicle exceeds 5°. Beyond 5° of slope, the maximum velocity of the vehicle is reduced linearly to values that, in the extreme, restrict the vehicle's velocity to 5 mph. The terrain ruggedness is fed into a controller with hysteresis that controls the velocity setpoint to exploit the linear relationship between filtered vertical acceleration amplitude and velocity; see Sec. 7.2. If rough terrain causes a vibration that exceeds the maximum allowable threshold, the maximum velocity is reduced linearly such that continuing to encounter similar terrain would yield vibrations which exactly meet the shock limit. Barring any further shocks, the velocity limit is slowly increased linearly with distance traveled.

This rule may appear odd, but it has great practical importance; it reduces the Stanley's speed when the vehicle hits a rut. Obviously, the speed reduction occurs after the rut is hit, not before. By slowly recovering speed, Stanley will approach nearby ruts at a much lower speed. As a result, Stanley tends to drive slowly in areas with many ruts, and only returns to the base trajectory speed when no ruts have been encountered for a while. While this approach does not avoid isolated ruts, we found it to be highly effective in avoiding many shocks that would otherwise harm the vehicle. Driving over wavy terrain can be just as hard on the vehicle as driving on ruts. In bumpy terrain, slowing down also changes the frequency at which the bumps pass, reducing the effect of resonance.

The velocity recommender is characterized by two parameters: The maximum allowable shock, and the linear recovery rate. Both are learned from human driving. More specifically, by recording the velocity profile of a human in rugged terrain, Stanley identifies The parameters that most closely match the human driving profile. Figure 23 shows the velocity profile of a human driver in a mountainous area of the 2004 Grand Challenge Course (the "Daggett Ridge"). It also shows the profile of Stanley's controller for the same data set. Both profiles tend to slow down in the same areas. Stanley's profile, however, is different in two ways: The robot decelerates much faster than a person, and its recovery is linear whereas the person's recovery is nonlinear. The fast acceleration is by design, to protect the vehicle from further impact.

Once the planner, velocity recommender, and health monitor have all submitted velocities, the minimum of these speeds is implemented by the velocity controller. The velocity controller treats the brake cylinder pressure and throttle level as two opposing single-acting actuators that exert a longitudinal force on the car. This is a very close approximation for the brake system, and was found to be an acceptable simplification of the throttle system. The controller computes a single error metric, equal to a weighted sum of the velocity error and the integral of the velocity error. The relative weighting determines the trade-off between disturbance rejection and overshoot. When the error metric is positive, the brake system commands a brake cylinder pressure

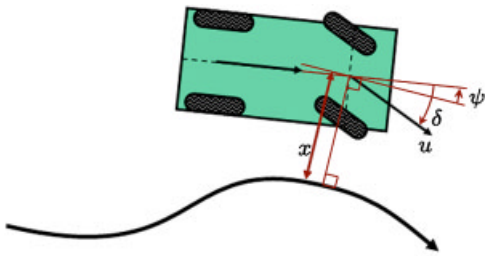


Figure 24. Illustration of the steering controller. With zero cross-track error, the basic implementation of the steering controller steers the front wheels parallel to the path. When cross-track error is perturbed from zero, it is nulled by commanding the steering according to a nonlinear feedback function.

proportional to the PI error metric; and when it is negative, the throttle level is set proportional to the negative of the PI error metric. By using the same PI error metric for both actuators, the system is able to avoid the chatter and dead bands associated with opposing single-acting actuators. To realize the commanded brake pressure, the hysteretic brake actuator is controlled through saturated proportional feedback on the brake pressure, as measured by the Touareg, and reported through the CAN bus interface.

9.2. Steering Control

The steering controller accepts as input the trajectory generated by the path planner, the UKF pose and velocity estimate, and the measured steering wheel angle. It outputs steering commands at a rate of 20 Hz. The function of this controller is to provide closed-loop tracking of the desired vehicle path, as determined by the path planner, on quickly varying potentially rough terrain.

The key error metric is the cross-track error, $x(t)$, as shown in Figure 24, which measures the lateral distance of the center of the vehicle's front wheels from the nearest point on the trajectory. The idea now is to command the steering by a control law that yields an $x(t)$ that converges to zero.

Stanley's steering controller, at the core, is based on a nonlinear feedback function of the cross-track error, for which exponential convergence can be shown. Denote the vehicle speed at time t by $u(t)$. In the error-free case, using this term, Stanley's front wheels match the global orientation of the trajectory. This is illustrated in Figure 24. The angle ψ in this diagram describes the orientation of the nearest path segment, measured relative to the vehicle's own orientation. In the absence of any lateral errors, the control law points the front wheels parallel to the planner trajectory.

The basic steering angle control law is given by

$$\delta(t) = \psi(t) + \arctan\frac{kx(t)}{u(t)}, \tag{7}$$

where k is a gain parameter. The second term adjusts the steering in (nonlinear) proportion to the cross-track error $x(t)$: The larger this error, the stronger the steering response toward the trajectory.

Using a linear bicycle model with infinite tire stiffness and tight steering limitations (see Gillespie, 1992) results in the following effect of the control law:

$$\dot{x}(t) = -u(t)\sin\arctan\left(\frac{kx(t)}{u(t)}\right) = \frac{-kx(t)}{\sqrt{1+\left(\frac{kx(t)}{u(t)}\right)^2}}, \tag{8}$$

and hence for small cross track error,

$$x(t) \approx x(0)\exp -kt. \tag{9}$$

Thus, the error converges exponentially to $x(t)=0$. The parameter k determines the rate of convergence. As cross-track error increases, the effect of the arctan function is to turn the front wheels to point straight toward the trajectory, yielding convergence limited only by the speed of the vehicle. For any value of $x(t)$, the differential equation converges monotonically to zero. Figure 25 shows phase portrait diagrams for Stanley's final controller in simulation, as a function of the error $x(t)$ and the orientation $\psi(t)$, including the effect of steering input saturation. These diagrams illustrate that the controller converges nicely for the full range attitudes and a wide range of cross-track errors, in the example of two different velocities.

This basic approach works well for lower speeds, and a variant of it can even be used for reverse driving. However, it neglects several important effects. There is a discrete variable time delay in

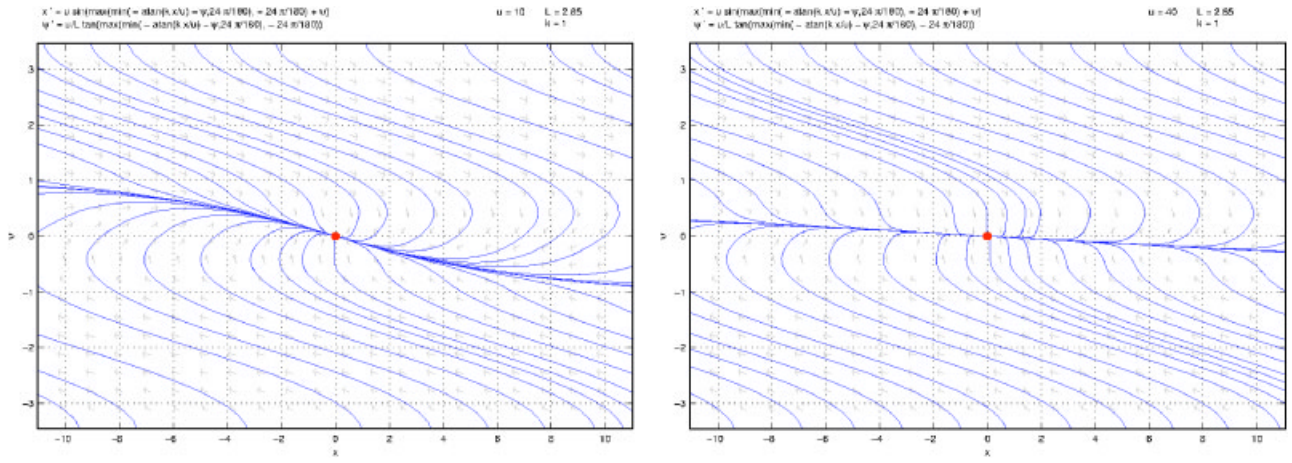


Figure 25. Phase portrait for $k=1$ at 10 and 40 m per second, respectively, for the basic controller, including the effect of steering input saturation.

the control loop, inertia in the steering column, and more energy to dissipate as speed increases. These effects are handled by simply damping the difference between steering command and the measured steering wheel angle, including a term for yaw damping. Finally, to compensate for the slip of the actual pneumatic tires, the vehicle is commanded to have a steady-state yaw offset that is a nonlinear function of the path curvature and the vehicle speed, based on a bicycle vehicle model, with slip, that was calibrated and verified in testing. These terms combine to stabilize the vehicle and drive the cross-track error to zero, when run on the physical vehicle. The resulting controller has proven stable in testing on terrain from pavement to deep off-road mud puddles, and on trajectories with tight enough radii of curvature to cause substantial slip. It typically demonstrates tracking error that is on the order of the estimation error of this system.

10. DEVELOPMENT PROCESS AND RACE RESULTS

10.1. Race Preparation

The race preparation took place at three different locations: Stanford University, the 2004 Grand Challenge Course between Barstow and Primm, and the Sonoran Desert near Phoenix, AZ. In the weeks leading up to the race, the team permanently moved to Arizona, where it enjoyed the hospitality of Volkswagen of America's Arizona Proving Grounds. Figure 26 shows examples of hardware testing in extreme offroad terrain; these pictures were taken while the vehicle was operated by a person.

In developing Stanley, the Stanford Racing Team adhered to a tight development and testing schedule, with clear milestones along the way. Emphasis was placed on early integration, so that an end-to-end prototype was available nearly a year before the race. The system was tested periodically in desert environments representative of the team's expectation for the Grand Challenge race. In the months leading up to the race, all software and hardware modules were debugged and subsequently frozen. The development of the system terminated well ahead of the race.

The primary measure of system capability was "MDBCF"—mean distance between catastrophic failures. A catastrophic failure was defined as a condition under which a human driver had to intervene. Common failures involved software problems, such as the one shown in Figure 9; occasional failures were caused by the hardware, e.g., the vehicle power system. In December 2004, the MDBCF was approximately 1 mile. It increased to 20 miles in July 2005. The last 418 miles before the National Qualification Event were free of failures; this included a

Figure 26. Vehicle testing at the Volkswagen Arizona Proving Grounds, manual driving.

single 200-mile run over a cyclic testing course. At that time, the system development was suspended, Stanley's lateral navigation accuracy was approximately 30 cm. The vehicle had logged more than 1,200 autonomous miles.

In preparing for this race, the team also tested sensors that were not deployed in the final race. Among them was an industrial strength stereo vision sensor with a 33 cm baseline. In early experiments, we found that the stereo system provided excellent results in the short range, but lagged behind the laser system in accuracy. The decision not to use stereo was simply based on the observation that it added little to the laser system. A larger baseline might have made the stereo more useful at longer ranges, but was unfortunately not available.

The second sensor that was not used in the race was the 24 GHz RADAR system. The RADAR uses a linear frequency shift keying modulated (LFMSK) transmit wave form; it is normally used for adaptive cruise control. After carefully tuning gains and acceptance thresholds of the sensor, the RADAR proved highly effective in detecting large frontal obstacles such as abandoned vehicles in desert terrain. Similar to the monovision system in Sec. 6, the RADAR was tasked to screen the road at a range beyond the laser sensors. If a potential obstacle was detected, the system limits Stanley's speed to 25 mph so that the lasers could detect the obstacle in time for collision avoidance.

While the RADAR system proved highly effective in testing, two reasons prevented its use in the race. The first reason was technical: During the NQE, the USB driver of the receiving computer repeatedly caused trouble, sometimes stalling the receiving computer. The second reason was pragmatical. During the NQE, it became apparent that the probability of encountering large frontal obstacles was small in high-speed zones; and even if those existed, the vision system would very likely detect them. As a consequence, the team felt that the technical risks associated with the RADAR system outweighed its benefits, and made the decision not to use RADAR in the race.

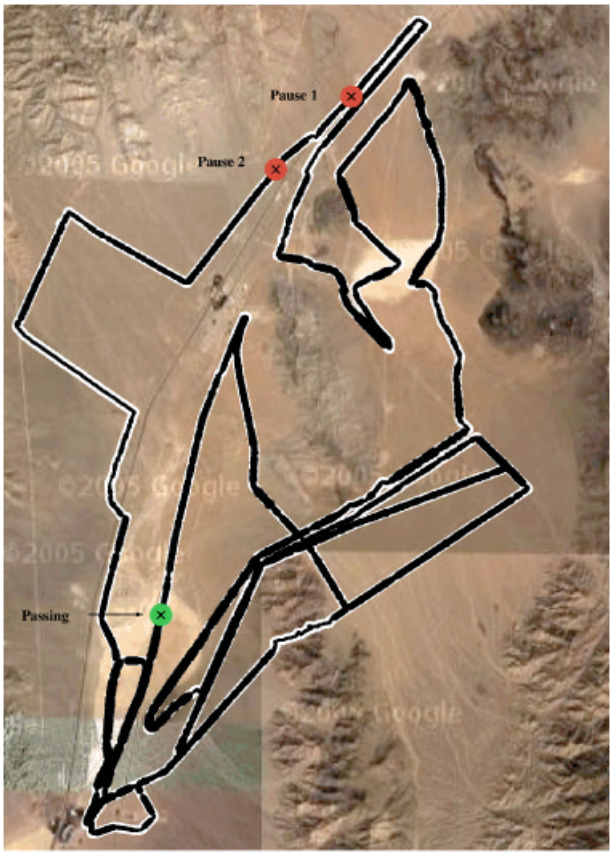


Figure 27. This map shows Stanley's path. The *thickness* of the trajectory indicates Stanley's speed (thicker means faster). At the locations marked by the red "x"s, the race organizers paused Stanley because of the close proximity of CMU's H1ghlander robot. At Mile 101.5, H1ghlander was paused and Stanley passed. This location is marked by a green "x".

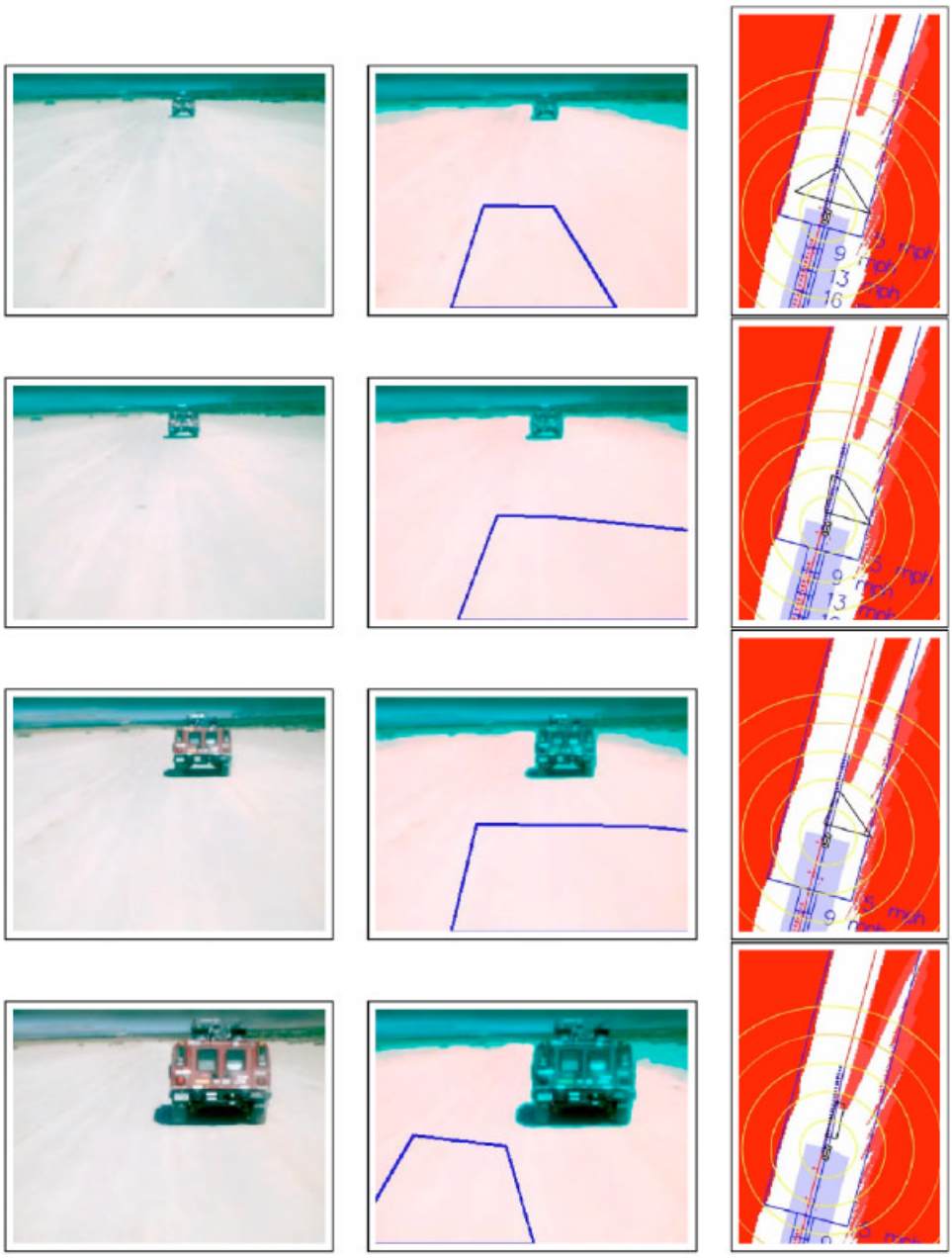


Figure 28. Passing CMU's H1ghlander robot: The left column shows a sequence of camera images, the center column the processed images with obstacle information overlayed; and the right column, the 2D map derived from the processed image. The vision routine detects H1ghlander as an obstacle at a 40 m range, approximately twice the range of the lasers.

10.2. National Qualification Event

The NQE took place from September 27 to October 5 on the California Speedway in Fontana, CA. Like most competitive robots, Stanley qualified after four test runs. From the 43 semifinalists, 11 completed the course in the first run, 13 in the second run, 18 in the third run, and 21 in the fourth run. Stanley's times were competitive, but not the fastest (Run 1—10:38; Run 2—9:12; Run 3—11:06; and Run 4—11:06). However, Stanley was the only vehicle that cleared all 50

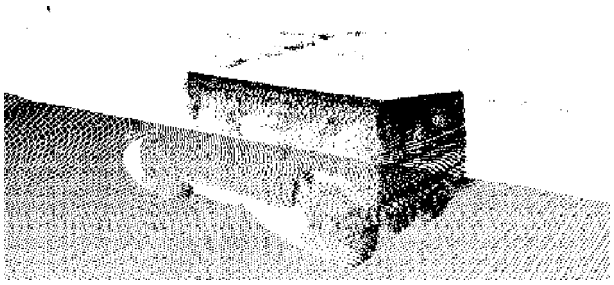


Figure 29. Laser model of CMU's H1ghlander robot, taken at Mile 101.5.

gates in every run, and avoided collisions with all of the obstacles. This flawless performance earned Stanley the number two starting position, behind CMU's H1ghlander robot and ahead of the slightly faster Sandstorm robot, also by CMU.

10.3. The Race

At approximately 4:10 am on October 8, 2005, the Stanford Racing Team received the race data, which consisted of 2,935 GPS-referenced coordinates, along with speed limits of up to 50 mph. Stanley started the race at 6:35 am on October 8, 2005. The robot immediately picked up speed and drove at or just below the speed limit. 3 h, 45 min, and 22 s into the race, at Mile 73.5, DARPA paused Stanley for the first time, to give more space to CMU's H1ghlander robot, which had started five minutes ahead of Stanley. The first pause lasted 2 min 45 s. Stanley was paused again only 5 min 40 s later, at Mile 74.9 (3 h, 53 min, and 47 s into the race). This time the pause lasted 6 min 35 s, for a total pause time of 9 min 20 s. The locations of the pauses are shown in Figure 27. From this point on, Stanley repeatedly approached H1ghlander within a few hundred yards. Even though Stanley was still behind H1ghlander, it was leading the race.

5 h, 24 min, 45 s into the race, DARPA finally paused H1ghlander and allowed Stanley to pass. The passing happened a Mile 101.5; the location is marked by a green circle in Figure 27. Figure 28 shows processed camera images of the passing process acquired by Stanley, and Figure 29 depicts a 3D model of H1ghlander as it is being passed. Since Stanley started in second pole position and finished first, the top-seeded H1ghlander robot was the only robot encountered by Stanley during the race.

As noted in the Introduction of this paper, Stanley finished first, at an unmatched finishing time of 6 h, 53 min, and 58 s. Its overall average velocity was 19.1 mph. However, Stanley's velocity varied wildly during the race. Initially, the terrain was flat and the speed limits allowed for much higher speeds. Stanley reached its top speed of 38.0 mph at Mile 5.43, 13 min and 47 s into the race. Its maximum *average* velocity during the race was 24.8 mph, which Stanley attained after 16 min and 56 s, at Mile 7.00. Speed limits then forced Stanley to slow down. Between Mile 84.9 and 88.1, DARPA restricted the maximum velocity to 10 mph. Shortly thereafter, at Mile 90.6 and 4 h, 57 min, and 7 s into the race, Stanley attained its minimum average velocity of 18.3 mph. The total profile of velocities is shown in Figure 30.

As explained in this paper, Stanley uses a number of strategies to determine the actual travel speed. During 68.2% of the course, Stanley's velocity was limited as precalculated, by following the DARPA speed limits or the maximum lateral acceleration constraints in turns. For the remaining 31.8%, Stanley chose to slow down dynamically, as the result of its sensor measurements. In 18.1%, the slow down was the result of rugged or steep terrain. The vision module caused Stanley to slow down to 25 mph for 13.1% of the total distance; however, without the vision module Stanley would have been forced to a 25 mph maximum speed, which would have resulted in a finishing time of approximately 7 h and 5 min, possibly behind CMU's Sandstorm robot. Finally, for 0.6% of the course, Stanley drove slower because it was denied GPS readings. Figure 31 illustrates the effect of terrain ruggedness on the overall velocity. The curve on the top illustrates the magnitude at which Stanley slowed down to accommodate rugged terrain; the bottom diagram shows the altitude profile, as provided by DARPA. The terrain ruggedness triggers mostly in mountainous terrain. We believe that the ability to adapt the speed to the ruggedness of the terrain was an essential ingredient in Stanley's success.

Stanley also encountered some unexpected difficulties along the course. Early on in the race, Stanley's laser data stream repeatedly stalled for durations of 300 to 1,100 ms. There were a total of 17 incidents, nearly all of which occurred between Mile 22 and Mile 35. The resulting inaccurate time stamp-

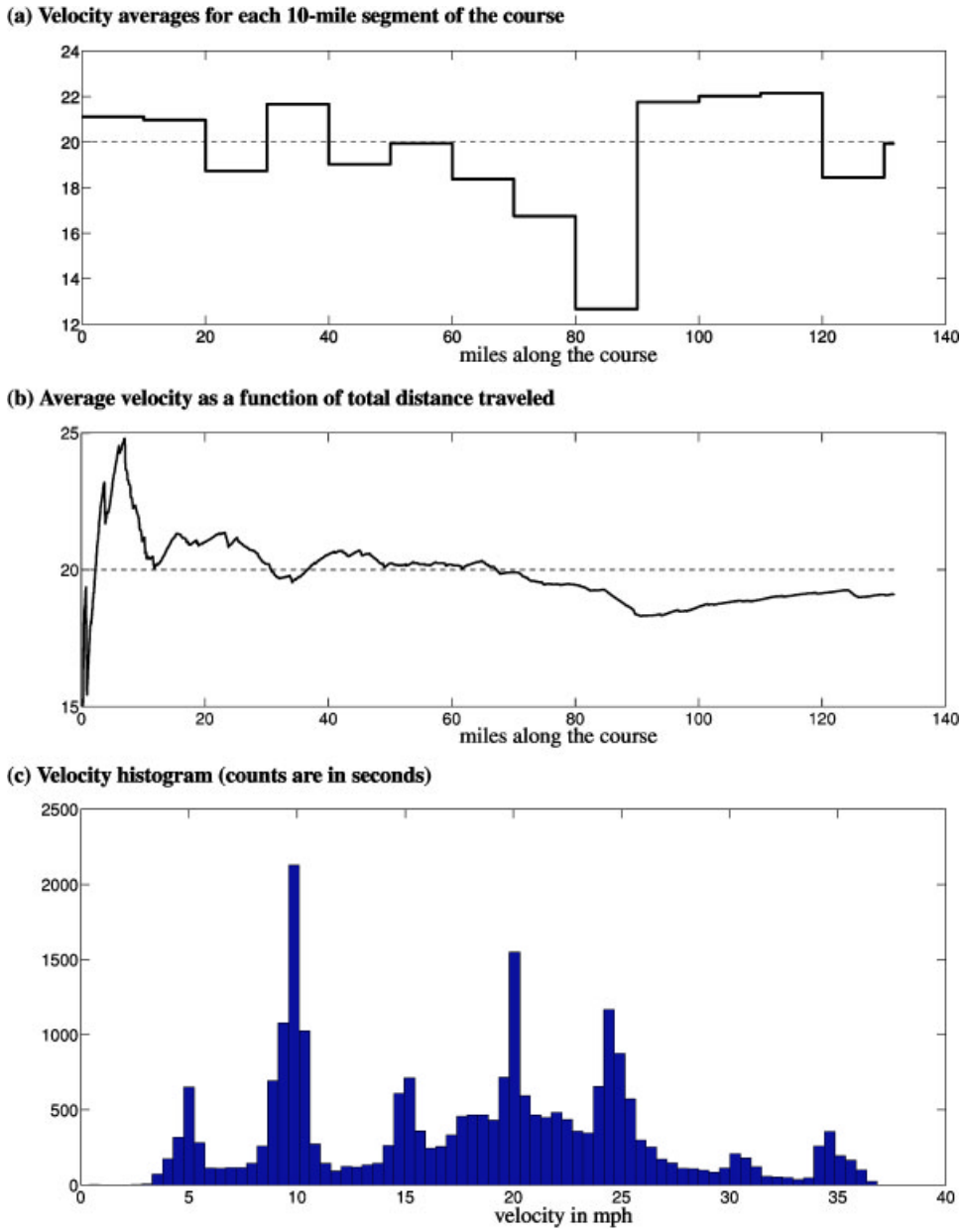


Figure 30. Stanley's cumulative velocity.

ing of the laser data led to the insertion of phantom obstacles into the map. In four of those cases, those incidents resulted in a significant swerve. The two most significant of these swerves are shown in Figure 32. Both of those swerves were quite noticeable. In one case, Stanley even drove briefly on the berm as shown in Figure 32(a); in the other, Stanley swerved on an open lake bed without any obstacles, as shown in Figure 32(b). At no point was the vehicle in jeopardy, as the berm that was traversed was drivable. However, as a result of these errors, Stanley slowed down a number of times between Miles 22 and 35. Thus, the main effect of these incidents was a loss of time early in the race. The data stream

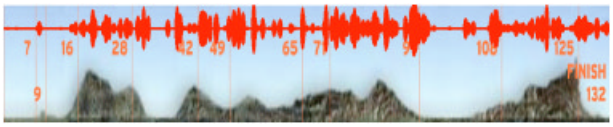


Figure 31. This diagram shows where the road conditions forced Stanley to slow down along the race course. Slow down predominately occurred in the mountains.

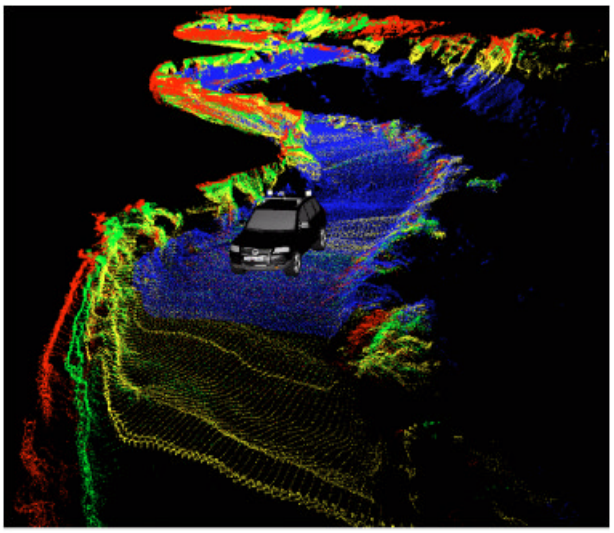


Figure 33. Sensor image from the Beer Bottle Pass, the most difficult passage of the DARPA Grand Challenge.

stalling problem vanished entirely after Mile 37.85. It only reoccurred once at Mile 120.66, without any visible change of the driving behavior.

During 4.7% of the Grand Challenge, the GPS reported 60 cm error or more. Naturally, this number represents the unit's own estimate, which may not necessarily be accurate. However, this raises the question of how important online mapping and path planning was in this race.

Stanley frequently moved away from the center axis of the RDDF. On average, the lateral offset was ±74 cm. The maximum lateral offset during the race was 10.7 m, which was the result of the swerve shown in Figure 32(c). However, such incidents were rare, and in nearly all cases nonzero lateral offsets were the results of obstacles in the robot's path.

An example situation is depicted in Figure 33. This figure shows raw laser data from the Beer Bottle Pass, the most difficult section of the course. An image of this pass is depicted in Figure 34(a). Of interest is the map in Figure 34(b). Here the DARPA-provided corridor is marked by the two solid blue lines. This image illustrates that the berm on Stanley's left reaches well into the corridor. Stanley drives as far left as the corridor constraint allows. Figure 35 shows a histogram of lateral offsets for the Beer Bottle Pass. On average, Stanley drove 66 cm to the right of the center of the RDDF in this part of the

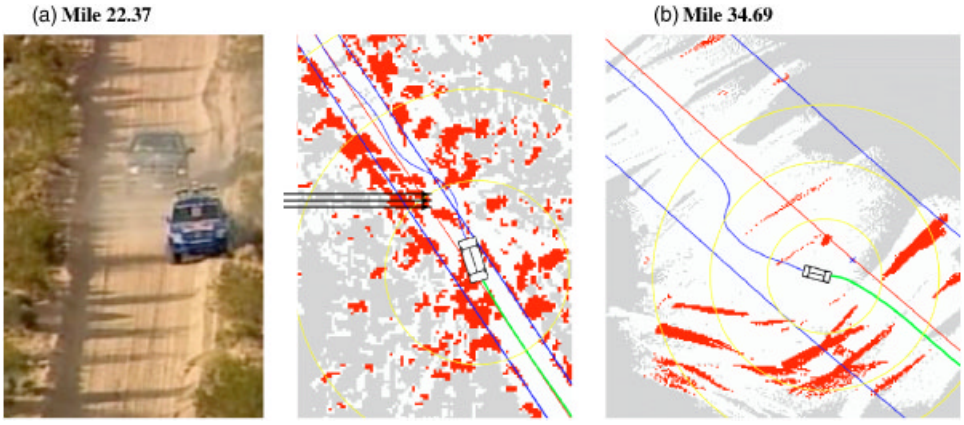


Figure 32. Problems during the race caused by a stalling of the laser data stream. In both cases, Stanley swerved around phantom obstacles; at Mile 22.37 Stanley drove on the berm. None of these incidents led to a collision or an unsafe driving situation during the race.

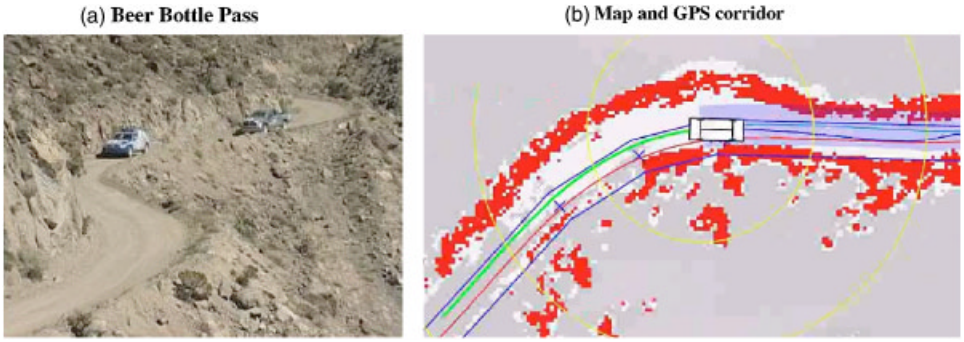


Figure 34. Image of the Beer Bottle pass, and snapshot of the map acquired by the robot. The two blue contours in the map mark the GPS corridor provided by DARPA, which aligns poorly with the map data. This analysis suggests that a robot that followed the GPS via points blindly would likely have failed to traverse this narrow mountain pass.

race. We suspect that driving 66 cm further to the left would have been fatal in many places. This sheds light on the importance of Stanley's ability to react to the environment in driving. Simply following the GPS points would likely have prevented Stanley from finishing this race.

11. DISCUSSION

This paper provides a comprehensive survey of the winning robot of the DARPA Grand Challenge. Stanley, developed by the Stanford Racing Team in collaboration with its primary supporters, relied on a software pipeline for processing sensor data and determining suitable steering, throttle, brake, and gear shifting commands.

From a broad perspective, Stanley's software mirrors common methodology in autonomous vehicle control. However, many of the individual modules relied on state-of-the-art artificial intelligence techniques. The pervasive use of machine learning, both ahead and during the race, made Stanley robust and precise. We believe that those techniques, along with the extensive testing that took place, contributed significantly to Stanley's success in this race.

While the DARPA Grand Challenge was a milestone in the quest for self-driving cars, it left open a number of important problems. Most important among those was the fact that the race environment was static. Stanley is unable to navigate in traffic. For autonomous cars to succeed, robots, such as Stanley, must be able to perceive and interact with moving traffic. While a number of systems have shown impressive results (Dickmanns et al., 1994; Hebert, Thorpe & Stentz, 1997; Pomerleau & Jochem, 1996), further research is needed to achieve the level of reliability necessary for this demanding task. Even within the domain of driving in static environments, Stanley's software can only handle limited types of obstacles. For example, the present software would be unable to distinguish tall grass from rocks, a research topic that has become highly popular in recent years (Dima & Hebert, 2005; Happold, Ollis & Johnson, 2006; Wellington, Courville & Stentz, 2005).

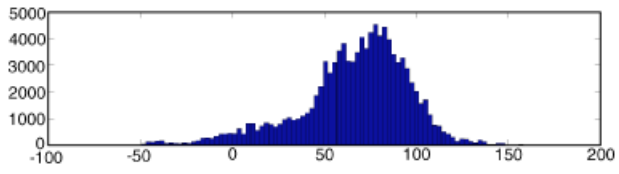


Figure 35. Histogram of lateral offsets on the beer bottle pass. The horizontal units are in centimeters.

ACKNOWLEDGMENTS

The Stanford Racing Team (SRT) was sponsored by four *Primary Supporters*: Volkswagen of America's Electronics Research Lab, Mohr Davidow Ventures, Android, and Red Bull. The Primary Supporters, together with the Stanford team leaders, formed the *SRT Steering Committee*, which oversaw the SRT operations. The SRT also received support from Intel

Research, Honeywell, Tyzx, Inc., and Coverity, Inc. Generous financial contributions were made by David Cheriton, the Johnson Family, and Vint Cerf. A huge number of individuals at Stanford, Volkswagen, and related facilities helped us in developing this robot, which is gratefully acknowledged.

The SRT also thanks DARPA for organizing this great race and the many journalists who provided press coverage. Our final gratitude goes to the many friends we made preparing for and during the event, and the many people who helped us and other teams along the way.

REFERENCES

Brooks, C., & Iagnemma, K. (2005). Vibration-based terrain classification for planetary exploration rovers. IEEE Transactions on Robotics, 21(6), 185–1191.

Crisman, J., & Thorpe, C. (1993). SCARF: A color vision system that tracks roads and intersections. IEEE Transactions on Robotics and Automation, 9(1), 49–58.

DARPA. (2004). DARPA Grand Challenge rulebook. Retrieved from http://www.darpa.mil/grandchallenge05/Rules_8oct04.pdf.

Davies, B., & Lienhart, R. (2006). Using CART to segment road images. In Proceedings SPIE Multimedia Content Analysis, Management, and Retrieval, San Jose, CA.

Dickmanns, E. (2002). Vision for ground vehicles: History and prospects. International Journal of Vehicle Autonomous Systems, 1(1), 1–44.

Dickmanns, E., Behringer, R., Dickmanns, D., Hildebrandt, T., Maurer, M., Schiehlen, J., & Thomanek, F. (1994). The seeing passenger car VaMoRs-P. In Proceedings of the International Symposium on Intelligent Vehicles, Paris, France.

Dima, C., & Hebert, M. (2005). Active learning for outdoor obstacle detection. In S. Thrun, G. Sukhatme, S. Schaal, & O. Brock (Eds.), Proceedings of the Robotics Science and Systems Conference, Cambridge, MA. Cambridge, MA: MIT Press.

Duda, R., & Hart, P. (1973). Pattern classification and scene analysis. New York: Wiley.

Ettinger, S., Nechyba, M., Ifju, P., & Waszak, M. (2003). Vision-guided flight stability and control for microair vehicles. Advanced Robotics, 17, 617–640.

Farrell, J., & Barth, M. (1999). The global positioning system. New York: McGraw-Hill.

Gat, E. (1998). Three-layered architectures. In D. Kortenkamp, R. Bonasso, & R. Murphy (Eds.), AI-based mobile robots: Case studies of successful robot systems (pp. 195–210). Cambridge, MA: MIT Press.

Gillespie, T. (1992). Fundamentals of vehicle dynamics. Warrendale, PA: SAE Publications.

Happold, M., Ollis, M., & Johnson, N. (2006). Enhancing supervised terrain classification with predictive unsupervised learning. In G. Sukhatme, S. Schaal, W. Burgard, & D. Fox (Eds.), Proceedings of the Robotics Science and Systems Conference, Philadelphia, PA. Cambridge, MA: MIT Press.

Hebert, M., Thorpe, C., & Stentz, A. (1997). Intelligent unmanned ground vehicles: Autonomous navigation research at Carnegie Mellon University. Dordrecht: Kluwer Academic.

Iagnemma, K., & Dubowsky, S. (2004). Mobile robots in rough terrain: Estimation, motion planning, and control with application to planetary rovers. Springer Tracts in Advanced Robotics (STAR) Series. Berlin: Springer.

Julier, S., & Uhlmann, J. (1997). A new extension of the Kalman filter to nonlinear systems. In International Symposium on Aerospace/Defense Sensing, Simulate and Controls, Orlando, FL.

Kelly, A., & Stentz, A. (1998). Rough terrain autonomous mobility, part 1: A theoretical analysis of requirements. Autonomous Robots, 5, 129–161.

Ko, N., & Simmons, R. (1998). The lane-curvature method for local obstacle avoidance. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Victoria, Canada. New York: IEEE Industrial Electronics Society.

Pomerleau, D.A. (1991). Rapidly adapting neural networks for autonomous navigation. In R.P. Lippmann, J.E. Moody, & D.S. Touretzky (Eds.), Advances in neural information processing systems 3 (pp. 429–435). San Mateo: Morgan Kaufmann.

Pomerleau, D.A. (1993). Knowledge-based training of artificial neural networks for autonomous robot driving. In J.H. Connell & S. Mahadevan (Eds.), Robot learning (pp. 19–43). New York: Kluwer Academic.

Pomerleau, D., & Jochem, T. (1996). Rapidly adapting machine vision for automated vehicle steering. IEEE Expert, 11(2), 19–27.

Simmons, R., & Apfelbaum, D. (1998). A task description language for robot control. In Proceedings of the Conference on Intelligent Robots and Systems (IROS), Victoria, CA. New York: IEEE Industrial Electronics Society.

van der Merwe, R. (2004). Sigma-point Kalman filters for probabilistic inference in dynamic state-space models. Unpublished Ph.D. thesis, Oregon Graduate Institute School of Science and Engineering, Beaverton, Oregon.

van der Merwe, R., & Wan, E. (2004). Sigma-point Kalman filters for integrated navigation. In Proceedings of the 60th Annual Meeting of The Institute of Navigation (ION), Dayton, OH.

Wellington, C., Courville, A., & Stentz, A. (2005). Interacting Markov random fields for simultaneous terrain modeling and obstacle detection. In S. Thrun, G. Sukhatme, S. Schaal & O. Brock (Eds.), Proceedings of the Robotics Science and Systems Conference, Cambridge, MA. Cambridge, MA: MIT Press.