Caroline: An Autonomously Driving Vehicle for Urban Environments

Fred W. Rauskolb
Herzfeld & Rubin, P.C.
40 Wall Street
New York, New York 10005

Kai Berger, Christian Lipski,
and Marcus Magnor
Institute of Computer Graphics
Mühlenpfordtstraβe 23
38106 Braunschweig, Germany

Karsten Cornelsen, Jan Effertz, Thomas Form,
Fabian Graefe, Sebastian Ohl,
Walter Schumacher, and Jörn-Marten Wille
Institute of Control Engineering
Hans-Sommer-Straβe 66
38106 Braunschweig, Germany

Peter Hecker and Tobias Nothdurft
Institute of Flight Guidance
Hermann-Blenk-Straβe 27
38108 Braunschweig, Germany

Michael Doering, Kai Homeier,
Johannes Morgenroth, and Lars Wolf
Institute of Operating Systems
and Computer Networks
Christian Basarke, Christian Berger, Tim
Gülke, Felix Klose, and Bernhard Rumpe
Institute of Software Systems Engineering
Mühlenpfordtstraβe 23
38106 Braunschweig, Germany
e-mail: carolo-uc@tu-bs.de

Received 29 January 2008; accepted 18 June 2008

Journal of Field Robotics 25(9), 674–724 (2008) © 2008 Wiley Periodicals, Inc.
Published online in Wiley InterScience (www.interscience.wiley.com). • DOI: 10.1002/rob.20254

The 2007 DARPA Urban Challenge afforded the golden opportunity for the Technische Universität Braunschweig to demonstrate its abilities to develop an autonomously driving vehicle to compete with the world's best. After several stages of qualification, our team CarOLO qualified early for the DARPA Urban Challenge Final Event and was among only 11 teams from initially 89 competitors to compete in the final. We had the ability to work together in a large group of experts, each contributing his expertise in his discipline, and significant organizational, financial, and technical support by local sponsors, who helped us to become the best non-U.S. team. In this report, we describe the 2007 DARPA Urban Challenge, our contribution, "Caroline," the technology, and algorithms, along with her performance in the DARPA Urban Challenge Final Event on November 3, 2007. © 2008 Wiley Periodicals, Inc.

1. MOTIVATION AND INTRODUCTION

Focused research is often centered around interesting challenges and awards. The airplane industry started off with awards for the first flight over the British Channel as well as the Atlantic Ocean. The Human Genome Project, the RoboCups, and the series of DARPA Grand Challenges for autonomous vehicles serve this very same purpose to foster research and development in a particular direction. The 2007 DARPA Urban Challenge took place to boost development of unmanned vehicles for urban areas. Although there is an obvious direct benefit for DARPA and the U.S. government, there will also be a large number of spin-offs in technologies, tools, and engineering techniques, both for autonomous vehicles and also for intelligent driver assistance. An intelligent driver assistance function needs to be able to understand the surroundings of the car, evaluate potential risks, and help the driver to behave correctly, safely, and, in case it is desired, also efficiently. These topics affect not only ordinary cars but also buses, trucks, convoys, taxis, and special-purpose vehicles in factories, airports, and more. It will take a number of years before we have a mass market for cars that actively and safely protect the passenger and the surroundings, such as pedestrians, from accidents in any situation.

Intelligent functions in vehicles are obviously complex systems. Large issues in this project were primarily the methods, techniques, and tools for the development of such a highly critical, reliable, and complex system. Adapting and combining methods from different engineering disciplines were important prerequisites for our success. For a stringent deadline-oriented development of such a system, it is necessary to rely on a clear structure of the project, a dedicated development process, and efficient engineering that fits the project's needs. Thus, we concentrated not only on the single software modules of our autonomously driving vehicle Caroline but also on the process itself. We furthermore needed an appropriate tool suite that allowed us to run the development and in particular the testing process as efficiently as possible. This includes a simulator allowing us to simulate traffic situations and therefore achieve a sufficient coverage of test situations that would have been hard to conduct in reality. Only a good collaboration between the participating disciplines allowed us to develop Caroline in time to achieve such a good result in the 2007 DARPA Urban Challenge.

In the long term, our goal was not only to participate in a competition but also to establish a sound basis for further research on how to enhance vehicle safety by implementing new technologies to provide vehicle users with reliable and robust driver assistance systems, e.g., by giving special attention to technology for sensor data fusion and robust and reliable system architectures, including new methods for simulation and testing. Therefore, the 2007 DARPA Urban Challenge provided a golden opportunity to combine expertise from several fields of science and engineering. For this purpose, the interdisciplinary team CarOLO had been founded, which drew its members from five different institutes. In addition, the team received support from a consortium of national and international companies.

In this paper, we first introduce the 2007 DARPA Urban Challenge and derive the basic requirements for the car from its rules in Section 2. Section 3 describes the overall architecture of the system, which is detailed in Section 4, describing sensor fusion, vision, artificial intelligence (AI), and vehicle control along with safety concepts. Section 5 describes the overall development process and discusses quality assurance and the simulator used to achieve sufficient

testing coverage in detail. Section 6 finally describes the evaluation of Caroline, namely the performance during the National Qualification Event and the DARPA Urban Challenge Final Event in Victorville, California, the results we found, and the conclusions to draw from our performance.

2. 2007 DARPA URBAN CHALLENGE

The 2007 DARPA Urban Challenge is the continuation of the well-known Grand Challenge events of 2004 and 2005, which were entitled "Barstow to Primm" and "Desert Classic." To continue the tradition of having names reflect the actual task, DARPA named the 2007 event "Urban Challenge," thus describing the nature of the mission to be accomplished.

The 2004 course, as shown in Figure 1, went from Barstow, California, to Primm, Nevada, and had a total length of about 142 miles. Prior to the main event, DARPA held a qualification, inspection, and demonstration for each robot. Nevertheless, none of the original 15 vehicles managed to come even close to the goal of successfully completing the course. With 7.4 miles (approximately 11.6 km) as the farthest distance traveled, the challenge ended very disappointingly and no one won the $1 million cash prize.

Thereafter, the DARPA program managers heightened the barriers for entering the 2005 challenge significantly. They also modified the entire quality inspection process to one involving a step-by-step application process, including a video of the car in action and the holding of so-called site visits, which involved the visit of DARPA officials to team-chosen test sites. The rules for these site visits were very strict, e.g., determining exactly how the courses had to be equipped and what obstacles had to be available. From initially 195 teams, 118 were selected for site visits, and 43 finally made it into the National Qualification Event (NQE) at the California Speedway in Ontario, California. The NQE consisted of several tasks to be completed and obstacles to be overcome autonomously by the participating vehicles, including tank traps, a tunnel, speed bumps, stationary cars to pass, and many more.

On October 5, 2005, DARPA announced the 23 teams that would participate in the final event. The course started in Primm, Nevada, where the 2004 challenge should have ended. With a total distance of 131.6 miles (approximately 211 km) and several natural obstacles, the course was by no means easier than the one from the year before. At the end, five teams completed it and the rest did significantly better than the teams the year before. The Stanford Racing Team was awarded the $2 million first prize.

In 2007, DARPA wanted to increase the difficulty of the requirements, in order to meet the goal set

Figure 1. 2004 DARPA Grand Challenge area between Barstow, California (A), and Primm, Nevada (B).

by Congress and the Department of Defense that by 2015 a third of the Army's ground combat vehicles would operate unmanned. Having already proved the feasibility of crossing a desert and overcoming natural obstacles without human intervention, now a tougher task had to be mastered. As the U.S. Armed Forces are currently facing serious challenges in urban environments, the choice of such seemed logical. DARPA used the good experience and knowledge gained from the first and second Grand Challenge events to define the tasks for the autonomous vehicles. The 2007 DARPA Urban Challenge took place in Victorville, California, as depicted in Figure 2.

The Technische Universität Braunschweig started in June 2006 as a newcomer in the 2007 DARPA Urban Challenge. Significantly supported by industrial partners, five institutes from the faculties of computer science and mechanical and electrical engineering equipped a 2006 Volkswagen Passat station wagon named Caroline to participate in the DARPA Urban Challenge as a "Track B" competitor.

Track B competitors did not receive any financial support from the DARPA. "Track A" teams had to submit technical proposals to get technology development funding awards up to $1,000,000 in fall 2006. Track B teams had to provide a 5-min video demonstrating the vehicle's capabilities in April 2007. Using these videos, DARPA selected 53 teams of the initial 89 teams that advanced to the next stage in the qualification process, the site visit, as conducted in the 2005 Grand Challenge.

Team CarOLO got an invitation for a site visit that had to take place in the United States. Therefore, team CarOLO gratefully accepted an offer from the Southwest Research Insitute in San Antonio, Texas, providing a location for the site visit. On June 20, Caroline proved that she was ready for the NQE in fall 2007. Against great odds, she showed her abilities to the DARPA officials when a huge thunderstorm hit San Antonio during the site visit. The tasks to complete included the correct handling of intersection precedence, passing of vehicles, lane keeping, and general safe behavior. After the demonstration, the team returned to Germany together with Caroline.

On August 9, the team received the results of the site visit event together with an invitation to the next stage of the qualification process, the NQE in Victorville, California. Being a semifinalist team, the team returned at the end of September to the Southwest Research Institute in San Antonio to finalize the development and tests. Three weeks

Figure 2. 2007 DARPA Grand Challenge Area in Victorville, California.

later, Caroline and the team arrived in Victorville, California, and participated in the NQE. To qualify for the Final Event, three courses had to be mastered by the vehicles, each one covering a certain part of the requirements. At the first course, called Track A, the robots needed to merge into moving traffic, Track B required the handling of very long and complex routes with stationary obstacles, and Track C tested intersections and how the vehicles handle the blockage of roads. Performing repeatedly in all tracks of the NQE, Caroline qualified early for the final stage, the DARPA Urban Challenge Final Event held on November 3. In Section 6, the overall performance of Caroline in the NQE and the DARPA Urban Challenge Final Event is illustrated.

3. SYSTEM ARCHITECTURE

Caroline is a standard 2006 Volkswagen Passat station wagon equipped with a variety of sensors, actuators, and computers to function as an autonomous mobile robot. In front, two multilevel laser scanners, one multibeam LIDAR sensor, and one radar sensor cover a field of view up to 200 m for approaching traffic or stationary obstacles. In addition, four cameras detect and track lane markings in order to allow precise lane keeping. The stereo vision system behind the windshield and another color camera combined with two laser scanners mounted on the roof were installed to provide information about the drivability of the terrain in front of the vehicle. Very similar to the front of the vehicle, one multilevel laser scanner, one medium-range radar, one LIDAR, and two radar-based blind spot detectors enable Caroline to detect obstacles at the rear. All these sensors are depicted in Figure 3.

An array of automotive personal computers (PC) mounted on a rack shown in Figure 4 functions as the hardware platform for a distributed software architecture with all internal communication based on Ethernet. The access to Caroline's by-wire steering, brake, and throttle system as well as to other low-level actuators is provided through a CANLOG III command interface, which also connects to the vehicle's E-stop system to provide emergency stop functionality even should the complete software system described below fail. Regardless of those lower level components described above, all computing and control hardware is based on industrial PC technology, thereby reducing hardware variety and simplifying failure management and component replacement.

The development of Caroline was divided among a number of institutes and disciplines, including faculties for computer science and mechanical and

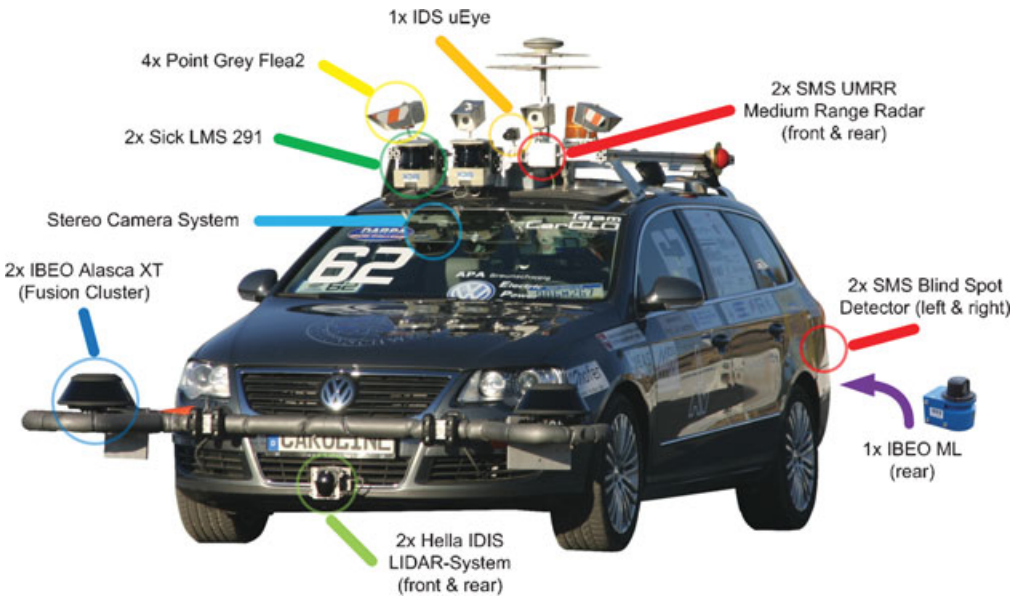


Figure 3. The perception system.

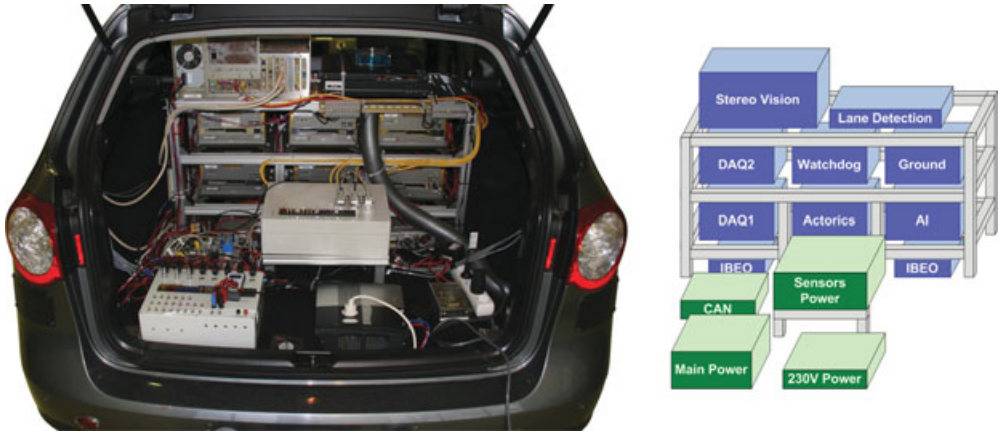


Figure 4. Computer rack and power supply.

electrical engineering. Mirroring this internal structure, Caroline's architecture is grouped into eight principal modules, interconnected with predefined interfaces as shown in Figure 5: sensor data acquisition, sensor data fusion, image processing, digital map, AI, vehicle path planning and low-level control, supervisory watchdog and online diagnosis, telemetry, and data storage for offline analysis. Owing to the intentionally linear signal flow between each function module without major signal loops, we were able to develop different modules independently and with minimum interference.

Starting at the bottom of this linear flow, the data acquisition unit provides necessary hard- and software modules to collect and process incoming data from all active sensors for object recognition. Because all of the sensors used are standard components originating from contemporary automotive driver assistance systems, they are equipped with a controller area network (CAN) communication interface. Taking into account the limitation of this bus standard regarding data throughput and determinism, a private sensor CAN was chosen for each sensor to keep latencies small and to avoid bus conflicts.

The acquisition of global positioning system (GPS) and inertial navigation system (INS) data (referred as ego state in the following) was moved directly into the real-time vehicle control unit in order to avoid large latencies within the closed-loop dynamic control. The time of day is obtained from the GPS and distributed via the network time protocol (NTP) to all subsystems.

Incoming video data are sampled from the assigned IEEE 1394 interface, preprocessed and interpreted directly on the image acquisition PCs to avoid overload of the vehicle's internal network by image data. Lane detection data are directly passed to the AI. The stereo vision system delivers three-dimensional (3D) scan points along with area data describing the drivability of the road. The data are fused with further scan points obtained from the laser scanners and area data from the additional color camera observing the ground in front of the vehicle. This fusion results in a drivability grid that is sent to the AI module.

Furthermore, following Caroline's signal flow, sensor data of all object-recognition sensors are processed within a central sensor data fusion unit as described in Section 4.1, which transmits the object-based vehicle's surroundings containing all static and dynamic targets in Caroline's field of view to the digital map. The digital map combines online environmental information with available offline information generated from mission definition files (MDFs) and route network definition files (RNDFs) provided for the mission. The combined data are the basis for the AI to generate driving decisions based on a distributed architecture for mobile navigation scheme (DAMN) as proposed by Rosenblatt (1997) and described in Section 4.3.

The driving commands obtained, e.g., "follow a given road," are issued to the soft real-time control module, which carries out trajectory generation and optimization based on driving dynamics of the

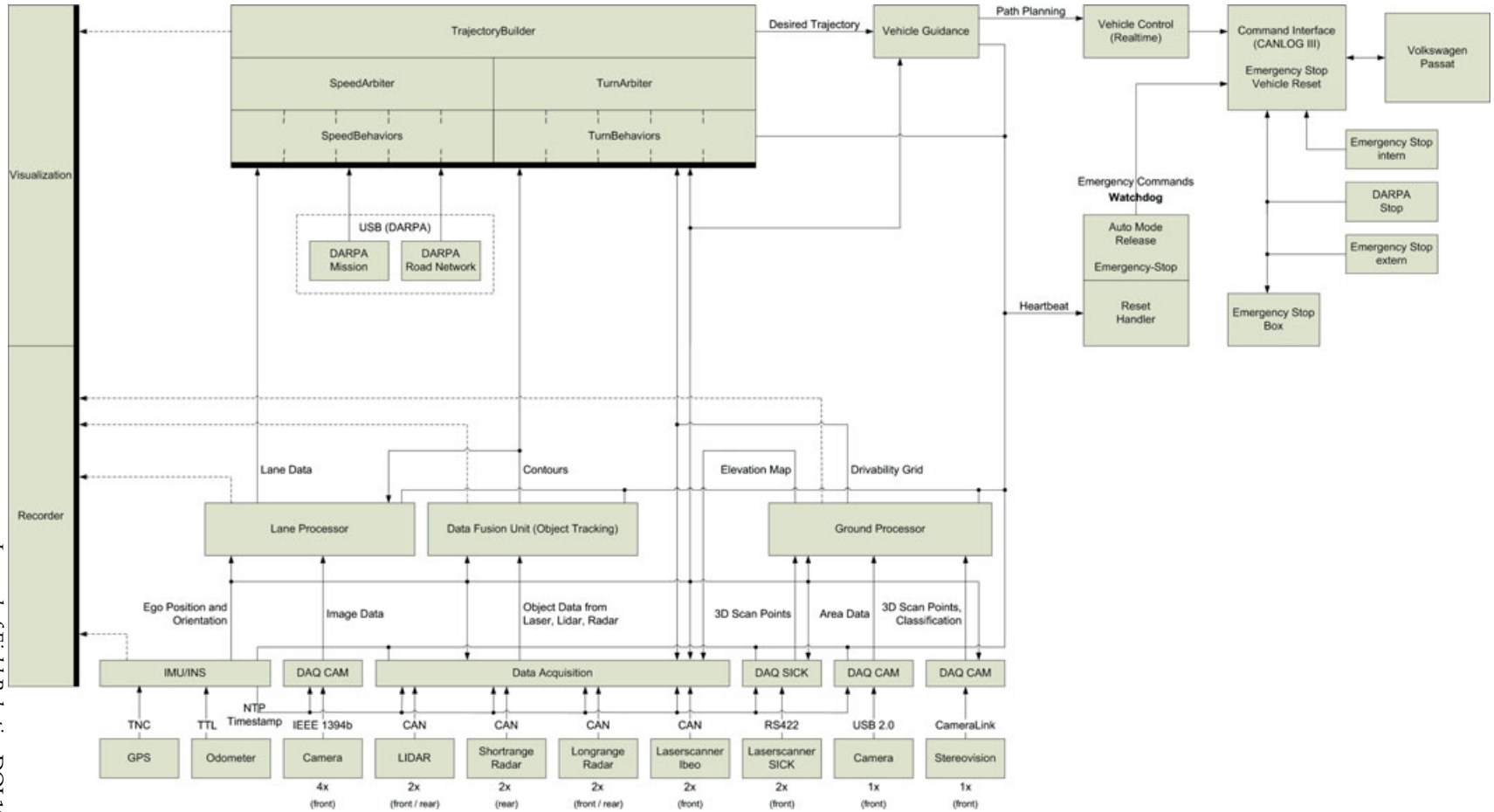


Figure 5. System architecture.

Journal of Field Robotics DOI 10.1002/rob

vehicle. The driving trajectories generated are then passed along into hard real-time control that addresses the vehicle actuators.

All modules previously described are supervised by a central watchdog process with the possibility to kill and restart one or several processes, computers, or sensors independently. Thus, a maximum of self-healing capability is installed in Caroline's systems.

The visualization module is used during development in order to display all exchanged object data. The data consist of, e.g., obstacles, lanes, terrain drivability, the planned path, and MDF. A recorder and a player module that logs data for the purpose of offline analysis are also integrated in this module.

4. SYSTEM MODULES

Caroline's software system consists of five modules. Tasks to be mastered in order to compete in the 2007 DARPA Urban Challenge are environment recognition, road finding, situation assessment, and vehicle control supervised by a safety module. These core modules are described below.

4.1. Sensor Fusion

Perception is one of Caroline's key systems. The system detects obstacles as well as the drivability of the environment. The sensor fusion system is separated in two parts. The first one is responsible for obstables, such as other cars, walls, or pedestrians. The other one takes care of the drivability of the environment. Thus, it is possible to keep the car on the road even in rough environments. Based on this information, the AI is able to find a safe path through traffic. The perception system will be described in greater detail in the following sections. The next section introduces the sensor concept, followed by the object-based data fusion and the grid-based fusion of the drivability.

4.1.1. Sensor Concept

A variety of sensor types originating from the field of driver assistance systems were chosen to provide detection of static and dynamic obstacles in the vehicle's surroundings as depicted in Figure 3:

- Below the license plate: A stationary-beam LIDAR sensor placed in the front and rear of the vehicle, with a range of approximately 200 m with an opening angle of 12 deg. The unit has an internal preprocessing stage and thus delivers its readings in an object-oriented fashion, providing target distance, target width, and relative target velocity with respect to the car's fixed sensor coordinate frame.
- White box below four cameras mounted on the roof: 24-GHz radar sensors were added to the front, rear, rear left, and right side of the vehicle. Whereas the center front and rear sensors provide a detection range of approximately 150 m with an opening angle of 40 deg, the rear right and left sensors function as blind-spot detectors with a range of 15 m and an opening angle of 140 deg due to their specific antenna structure. The front sensor acts as a stand-alone unit delivering object-oriented target data, such as position and velocity, through its assigned external control unit (ECU). The three radar sensors in the rear section operate as a combined sensor cluster using an additional ECU, providing object-oriented target data in the same fashion as the front system. From the perspective of the postprocessing fusion system, the three rear sensors can therefore be regarded as one unit.
- Laser scanner mounted on the left- and right-hand sides: Two Ibeo ALASCA XT laser scanners were installed in the vehicle's front section, each providing an opening angle of 240 deg with a detection range of approximately 60 m. The raw measurement data of both front laser scanners are preprocessed and fused on their corresponding ECU, delivering complex object-oriented target descriptions consisting of target contour information, target velocity, and additional classification information. Additionally, the raw scan data of both laser scanners can be read by the fusion system's grid-based subsection.
- One Ibeo ML laser scanner was added to the rear side, providing similar detection capabilities as the two front sensors, with a reduced opening angle of 180 deg due to its mounting position. All Ibeo sensors are based on a four-plane scanning principle with a vertical opening angle of 3.2 deg between the top and bottom scan plane. This opening angle enables smaller pitch movements of the vehicle to be covered.

- Two SICK LMS-291 laser scanners were mounted on the vehicle's front roof section. These scanners are based on a single-plane technology. They were set to measure the terrain profile at 10 and 20 m, respectively. The view angle was limited by software to 120 deg.
- A stereo vision system mounted behind the vehicle's front window covers an area of approximately 60 deg within a range of 50 m, providing 3D terrain profile data for all stereo vision points retrieved. A simple classification into the driveway, curb, and obstacles classes is also available.
- Between the four cameras on the roof: A universal serial bus (USB)–based color mono camera installed on the front roof section, covering an opening angle of approximately 60 deg.

The sensors' view areas are shown in Figure 6. These view areas overlap for a more reliable view of the environment.

The sensor architecture described reflects the hybrid postprocessing scheme applied. Whereas the first four sensors deliver their data in an object-oriented fashion and are therefore treated within the system's object tracking and data fusion stage, the three last sensors described are evaluated based on their raw measurement data in the grid-based subsection. A distributed data fusion system consisting of three interconnected units was set up. To equally balance the available computing power, the object tracking system was split into two independent modules, covering the front and rear sections independently. Therefore, two automotive computers carry out data acquisition and data fusion of the front and rear object detecting sensors, and the third PC is used to fuse the raw sensor readings of the SICK scanners, stereo vision system, and mono color camera as shown in Figure 7.

4.1.2. Object Tracking Fusion

The object fusion system is based on a pipes-and-filters pattern as depicted in Figure 8. All incoming sensor data are queued and then processed sequentially using a first-in-first-out strategy. Within the first step, data association is carried out in order to assign incoming sensor objects to their corresponding tracks in the fusion system that are taken from a real-time track database. In case of a positive match between an existing track and incoming sensor object,

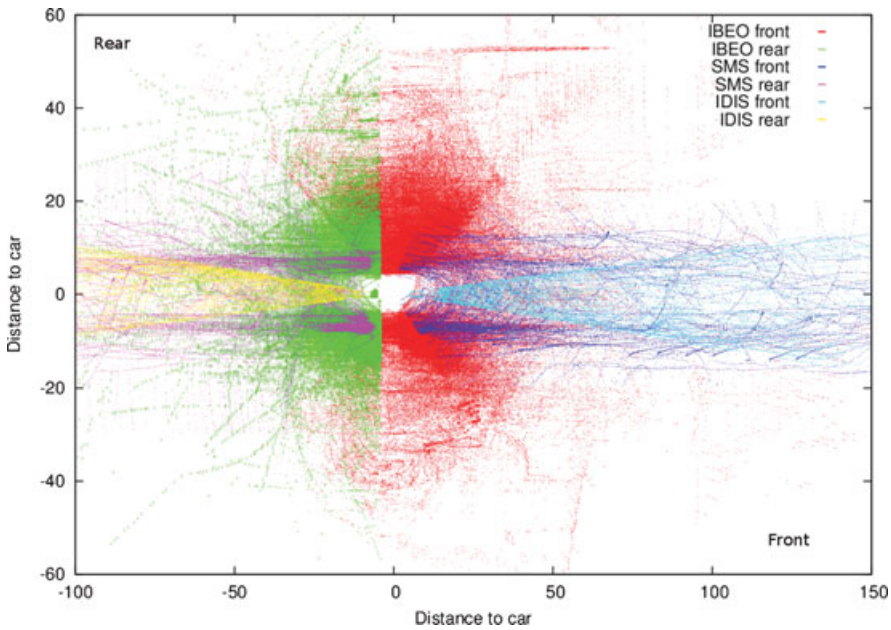


Figure 6. Sensor view areas.

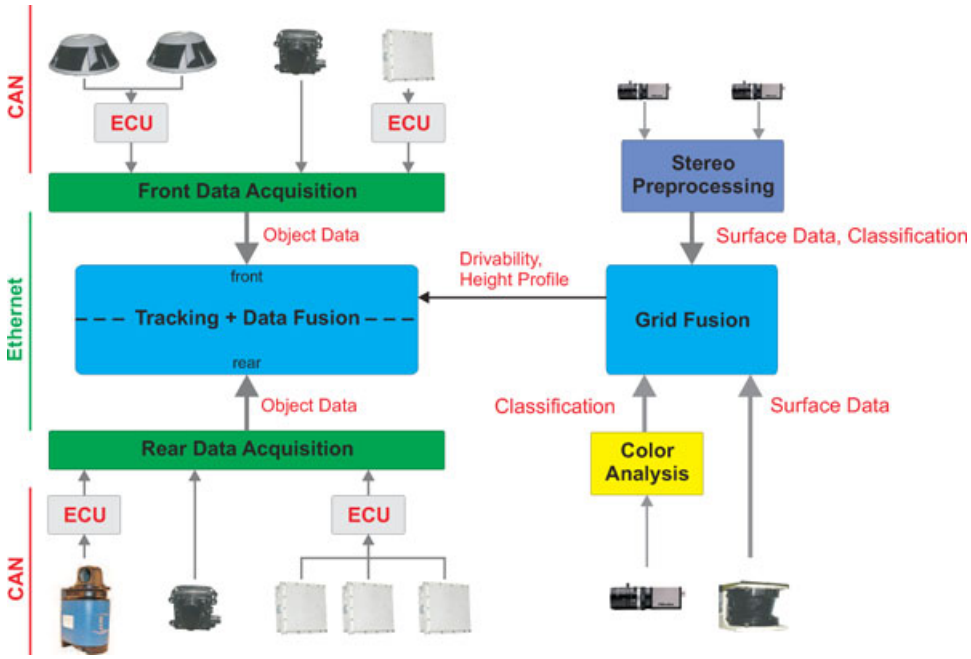


Figure 7. Fusion architecture.

this pairing is then pushed into the processing queue of the system's extended Kalman filter (EKF) in order to correct the track with new measurement data. If no match can be found, the sensor object is regarded as a potential new target and pushed into the pretracking system. Within pretracking, sensor data are justified against time and all other sensors, taking into account sensor redundancy when applicable. Pretracking and data association will be described later in greater detail.

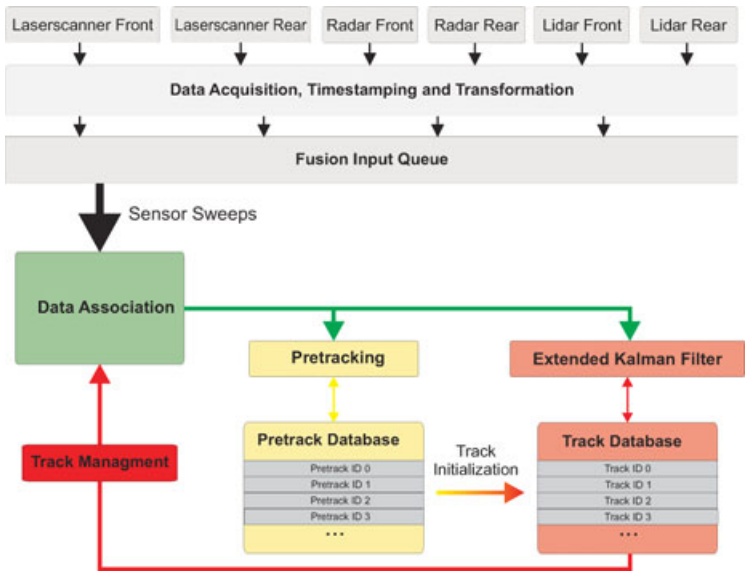


Figure 8. Object fusion system architecture.

If a sensor object has reached a certain level of justification, a new track will be instantiated and pushed into the real-time track database. Parallel to data association, pretracking, and final object tracking, a track management unit periodically scans the track database for "dead" tracks, i.e., fusion objects that have not been updated for a certain amount of time. In addition to this garbage collection, all valid tracks are compared to each other for track merging and track splitting, which is necessary to handle situations including a passenger entering or leaving his vehicle or any other situation in which two objects in the real world converge or split. Instead of transferring a whole-track database image to downstream modules, create, update, and delete messages of the track database are issued via the network. Every client is then capable of maintaining its own track database. Therefore, network load can be significantly reduced without any loss of information.

Data association and pretracking Data association and pretracking have a key functionality within Caroline's fusion system. Imperfect data association leads inevitably to incorrect tracks, whereas incorrect track initialization during pretracking leads to imperfect data association, because correct tracks and false alarms will then compete for incoming measurement data. With this central position, the association and pretracking stage dominates the state estimator in the main tracking stage, because no state estimator can transform falsely associated sensor readings into useful update information for a track. In classical tracking approaches in which objects are mostly described through a state vector consisting of a generalized object position, velocity, and, if applicable, further derivatives of these quantities, data association can be performed in a point-to-point matching process.

Within Caroline's fusion system, these approaches had to be extended in order to handle spread objects with complex shapes. Three different types of sensor objects have to be processed: complex contours delivered by laser scanners, line-shaped objects delivered by the LIDAR system, and classical point-shaped objects received from radar sensors. It is not possible to define a common general object position seen by all sensors, because each sensor will most likely see the target differently. For example, the point of reflection delivered from a radar is unknown compared to precise contour measurements gained from a laser scanner. Additionally, as the vehicle moves through the real world, the point of reflection of each sensor type moves on the outline of a real-world object. Therefore a multipoint track model was chosen, describing a detected object by an arbitrary number of contour points and postulating a common movement vector following a rigid-body assumption. This way each contour measurement can be matched to the tracked contour point with the best fit. A two-staged data association process was set up, with the first stage serving as a justification as to whether track and measurement describe the same real-world object and the second stage calculating the optimal contour association between measured and tracked object points. Within stage 1, a weighting function counting for the minimum Euclidean distance and similarity of velocities is calculated:

$$w_{i,j} = a \cdot \min\left[\left|x_k^i - x_l^j\right|, \forall k, l\right] + b \cdot |v_i - v_j|, \qquad (1)$$

with $w_{i,j}$ being a scalar weight for association between track i and measurement j, tracked and measured velocity vectors v_i, v_j, x_k^i, x_l^j being the kth and lth contour point position of track i, and measurement j and a, b serving as tuning parameters. A threshold for this weight is further defined and an association below that threshold level will be pushed into stage 2.

In stage 2, an optimal match between all measured and tracked contour points is calculated based on an association matrix:

$$\Omega = \begin{bmatrix} \left|x_1^i - x_1^j\right| & \dots & \left|x_1^i - x_l^j\right| \\ \dots & \dots & \dots \\ \left|x_k^i - x_1^j\right| & \dots & \left|x_k^i - x_l^j\right| \end{bmatrix}. \qquad (2)$$

Optimization can be carried out with standard algorithms such as the Hungarian/Munkres method, nearest-neighbor, or similar approaches. We used the fast minimum algorithm. This two-staged association process avoids unnecessary computational load on the system, because unlikely associations will be filtered out in stage 1 while the computational challenging minimization is carried out only for positive matches.

During pretracking, incoming sensor data are first associated with preliminary track objects (pretracks) using methods similar to those described above. A pretrack carries along a vector of sensor assignments, storing for each sensor type the last assigned sensor object ID. A simple Kalman filter based on a constant velocity motion model is calculated for

each pretrack to update its position given by incoming sensor data. In addition to the vector of sensor assignments, an update counter is carried along, storing the number of positive association events. Taking into account sensor redundancies read from a configuration file, a threshold for track activation is evaluated based on this update counter, depending on the level of redundancy in the affected observation area of that object. A simple description language was implemented to efficiently model these redundancies and to influence the update count threshold for track activation, e.g.,

```
polygon={0,2;10,2;10,-2;0,-2}
modifyCount=2000
condition=( RADARFront &&
          !( LASERFront || LIDARFront ) ),
```

which means, for the fusion system, "activate track in a 2 × 10 m, box-shaped view area after 2,000 positive matches when it is seen only by the front radar system and not by the laser scanners or LIDAR sensors," which, in this case, serves as protection against random, unstable false alarms from the radar sensor directly in front of the vehicle.

Tracking and data fusion For the main tracking algorithm, a model-switching EKF based on two-track motion models was implemented. A six-dimensional (6D) motion model describes fast-moving objects using a state vector:

$$x^{6D} = \begin{pmatrix} x_{1\ldots n} \\ y_{1\ldots n} \\ v \\ a \\ \alpha \\ \omega \end{pmatrix}, \tag{3}$$

with $x_{1\ldots n}$ and $y_{1\ldots n}$ being the x and y coordinates of the n contour points, the common velocity, acceleration, course angle, and course angle velocity with respect to the global Earth-fixed reference frame. For slow or static objects, a simpler four-dimensional (4D) state vector was chosen:

$$x^{4D} = \begin{pmatrix} x_{1\ldots n} \\ y_{1\ldots n} \\ v \\ a \end{pmatrix}, \tag{4}$$

thus taking into account that the majority of detected objects are of a rather static nature and distribution of available sensor information in many unnecessary state variables is suboptimal in that case. As seen in Eqs. (4) and (5), the classical state vector has been enriched by the number of contour points, thus making it necessary to extend the Kalman filter algorithm (Kalman, 1960) to handle multiple positions within the same state vector. Similarly, we define the sensor measurement vector for a sensor object consisting of m contour points:

$$y = \begin{pmatrix} x_{1\ldots m} \\ y_{1\ldots m} \\ v_x \\ v_y \end{pmatrix}, \tag{5}$$

with x_1, y_1, v_x, v_y being measured contour point x and y coordinates as well as x- and y-velocity components with respect to the global Earth-fixed reference frame. Postulating a common position noise covariance for all contour points within track and measurement, the update algorithm can be extended as follows:

$$\begin{aligned} x_k(v+1|v) &= f(x_k(v)), \\ P(v+1|v) &= F^T \cdot P \cdot F + Q, \\ s_{k,l} &= y_l(v+1) - h(x_k(v+1|v)), \\ S(v+1) &= H \cdot P(v+1|v) \cdot H^T + R, \\ K(v+1) &= P(v+1|v) \cdot H^T \cdot S(v+1)^{-1}, \\ r_{k,l}(v+1) &= K(v+1) \cdot s_{k,l}(v+1), \end{aligned} \tag{6}$$

with x_k being the track state vector regarding contour point k, $f(x)$ the nonlinear system transfer function, P the common state covariance matrix, F the system transfer Jacobian, Q the system noise covariance, $s_{k,l}$ the innovation vector of tracked contour point k compared with measured point l of the associated sensor object, y_l the sensor measurement vector regarding measured point l, $h(x)$ the nonlinear system output function, S the common innovation covariance matrix, H the system output Jacobian, R the estimated measurement noise, K the Kalman gain in this update cycle, and $r_{k,l}$ the correction vector for tracked contour point k getting updated with measured point l.

The tracked contour points can then be updated by adding the first two components of the associated vector $r_{k,l}$. To calculate updated common velocity,

acceleration, course angle, and course angle velocity in the 6D movement model, the mean value for vector $r_{k,l}$ is calculated over all given contour point associations:

$$r_{\text{mean}} = \frac{1}{N} \sum_{k,l=1}^{N} r_{k,l}, \tag{7}$$

with N being the total number of acquired contour point matches within the second stage of data association. Corrected common values can then be acquired by adding the last four components of vector r_{mean} to the corresponding elements in the track state vector.

Obviously, by postulating a common system and measurement noise covariance for all contour points, Kalman gain can be computed once per update cycle. Although it would theoretically be possible to calculate a separate Kalman gain for each tracked contour point and therefore remove the limitations to system and measurement covariance, this would lead to an N-times bigger computational load, because matrix inversion of the system innovation covariance matrix is the most costly part of the algorithm. In this case, the algorithm would simply calculate a separate Kalman filter for each contour point, which is not practically realizable in a real-time application. In the approach described we have no significantly higher computational effort compared to a standard EKF, while at the same time realizing spread-contour functionality and removing the need for a stable point of reference for tracked objects.

To prevent the track from being flooded with contour points, a garbage collection mechanism was installed by carrying along update counters for each contour point, which stores the last update time stamp and the overall number of updates counted to that point in time. In this manner, inactive contour points can be detected easily and removed from the track's point list.

Object splitting and termination Because of the track's polyline object model, it is necessary to implement a track splitting algorithm. If there is no such method, one track can collect points from many objects and grow to a rather huge but meaningless track. For example, a person dropping off a car and moveing away would still be part of the car track because of the data association algorithm depicted in Figure 9. When the person just dropped off and is still near the car, it will become one track. After the person moves away from the car, the contour points will still be updated because there is an object at the position of the car and the person is also still there. Between these two objects there is nothing but the polyline from the track still describing an outline of an object.

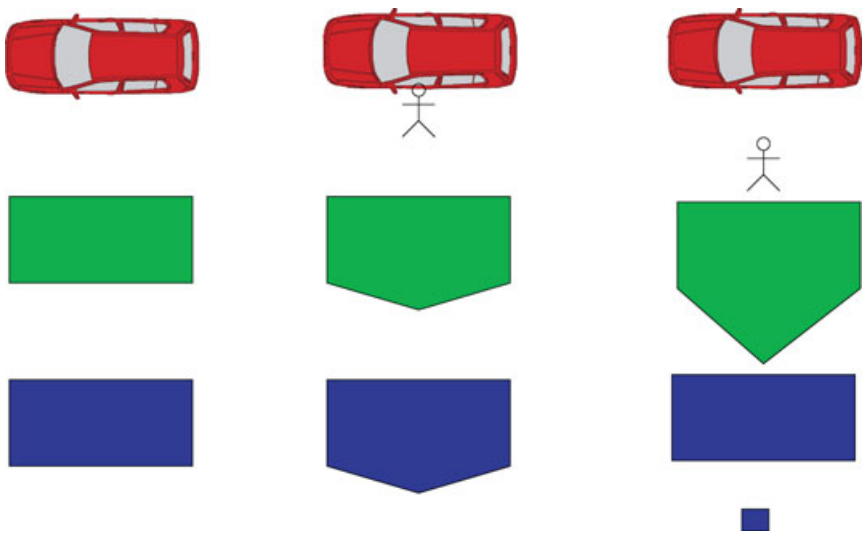


Figure 9. Person who drops off a car. From left to right: Person still in the car, person just dropped off, person moves away. From top to bottom: Reality, track without splitting, track after splitting.

To detect these false tracks, an algorithm was developed to split such tracks. The basic idea is to examine the objects based on the raw sensor object data and find indepentent sets of objects. These independent sets will become the new tracks. Normally, there are no such sets, but in the event of an unsplit track, there are two or more partitions. Polygonal objects around the track will be described as a planar undirected two-colored graph. The algorithm contains the following steps:

1. Build planar undirected colorable rectangular graph. Set the color of every node to black.
2. Set the polylines of every sensor object of the track to white.
3. Search for independent sets in the graph (Cormen, Leiserson, Rivest, & Stein, 2002).
4. If more than one set is found, build new tracks from the points describing the outlines.

The algorithm runs periodically during track garbage collection. Although complexity depends on the maximal area a covered by a track $O(a)$, this algorithm can be implemented efficiently with graphic libraries.

4.1.3. Grid-Based fusion

In contrast to the object tracking subsystem, the grid fusion system does not describe agents in the vehicle's environment with discrete state vectors but instead discretizes the whole environment into a rectangular matrix (grid) structure. Each grid cell carries a number of assigned features:

- a height value expressed in the global Earth-fixed reference frame
- a gradient value describing the height difference to neighboring cells
- a set of Dempster–Shafer probability masses counting for the hypotheses undrivable, drivable, and unknown
- a status flag stating whether measurement data have been stored within the corresponding cell
- an update time stamp storing the last time a cell update was carried out

Data structure The biggest challenge with grid-based models in an automotive environment is the need for real-time operation. High maneuvering speeds in automotive applications require update rates greater than 10 Hz, which is almost too low because this equals a travel distance of 1.4 m at normal urban speeds. The approach of discretizing the environment into grid cells leads to significantly high memory requirements and therefore calls for efficient data structures. As an example, the storage of a view area of only 100 × 100 m with a resolution of 25 cm generates 160,000 grid cells. Assuming a four-byte floating point value for each feature as described above, this grid extends up to 3 Mbyte. Together with an update rate of 10 Hz, this leads to a constant data throughput of 256 Mbit/s, which in any case is more than the standard automotive bus infrastructure would be able to handle. Efficient algorithms and data reduction prior to serialization is therefore the key to a successful application. For addressing these issues we implemented a rolling grid data structure wherein the vehicle's own position is a pointer to the corresponding grid cell. This position will be regarded as a virtual origin for all incoming sensor readings, which can then be subsequently accessed by moving through the double-linked data structure relative to that virtual origin. The main grid is again subdivided into subgrids whose sizes match the processor's caching mechanism for optimal usage of the available computing resources. Although it would theoretically be possible to make the surface large enough to cover the expected maneuvering area, this would lead to extremely high memory usage and is therefore not feasible. Instead, when the vehicle moves through the world, the reference point shifts along the double-linked spherical list. As soon as it crosses the border from one subgrid to the next, the corresponding subgrids at the new horizon of the data structure are cleared and are therefore available for new data storage.

Treatment of laser and stereo vision point data As the first step within grid data fusion, the 3D point clouds received from the laser scanners and stereo vision system are transformed into the global Earth-fixed reference frame taking vehicle attitude into account (roll, pitch, and yaw) as acquired from the GPS/INS unit and sensor-specific calibration information. The accuracy of these transformations is crucial to subsequent postprocessing. Vehicle height as delivered by the GPS is especially important and is therefore subject to further filtering and justification. For each measured point, the corresponding grid cell is retrieved and a ray-tracing algorithm (Bresenham) is carried out to update all cells from the sensor

coordinate system's origin to the measured target point. Several versions of the Bresenham algorithm are described in the literature; in this case we will introduce the 2D version following Pitteway (1967) for reasons of simplicity.

The lines are traced similar to the functionality of a plotter, which is basically the origin of that algorithm. On the way through the traced lines, each cell passed is updated according to following rules:

- If the cell lies on the path between sensor origin and measured target point and its height value exceeds the current Bresenham line height value, reduce the stored value to that of the current Bresenham line.
- If the cell is the end point of the Bresenham line, store the associated height value.
- In both cases, store update time stamp and mark that cell as having been measured.

The grid is updated following the direct optical travel path of any laser ray (or virtual stereo vision ray) starting at the sensor origin and ending at the target point as depicted in Figure 10. This model follows the assumption that any obstacle would block the passing optical ray and therefore any cells on the traveling path must be lower than the ray itself.

Data fusion Parallel to entering the 3D point, data acquired from laser scanners and stereo vision, vision-based classification are processed using a Dempster–Shafer approach (Shafer, 1976, 1990). A sensor model was created for each data source, mapping the sensor-specific classification into the Dempster–Shafer probability mass set, which can then be fused into the existing cell probability masses using Dempster's rule of combination:

$$m_c^*(A) = m_c(A) \bigoplus m_m(A) = \frac{1}{1-K} \sum_{B \cap C = A \neq \emptyset} m_c(B) m_m(C), \tag{8}$$

with m_c, m_m being the cell and measurement probability mass set and m_c^* the combined new set of masses for the regarded cell, whereas the placeholders A, B, and C can describe any of the three hypotheses drivable, undrivable, and unknown. The term K expresses the amount of conflict between existing cell data and incoming measurement, with

$$K = \sum_{B \cap C = \emptyset} m_c(B) m_m(C). \tag{9}$$

The mass set m_m has to be modeled out of the acquired sensor data.

With respect to the stereo vision system, which is capable of classifying retrieved point clouds into the classes road, curb, and obstacle, this mapping is trivial and can be done by assigning an appropriate constant mass set to each classification result. The exact values of these masses can then be subject to further tuning in order to trim the fusion system for maximum performance given real sensor data.

In the case of the mono vision system, Caroline assigns each pixel in the retrieved image a drivability value P_d between 0.0 representing undrivable and 1.0 representing fully drivable; a mapping function is then applied, which creates the three desired mass values D, drivable; U, undrivable; and N, unknown,

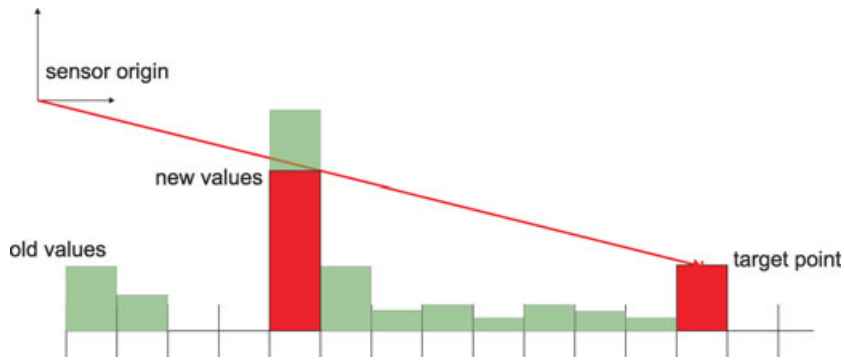


Figure 10. Ray update mechanism.

that can be either drivable or undrivable as follows:

$$
\begin{aligned}
m_m(D) &= D_{\max} \cdot P_d, \\
m_m(N) &= (1 - D_{\max}), \\
m_m(U) &= 1 - m_m(D) - m_m(N).
\end{aligned} \tag{10}
$$

The value $D_{\max}$ will serve as a tuning parameter, influencing the maximum trust placed into the mono vision system and based on the quality of its incoming data. The classification mechanism of the stereo vision system is beyond the scope of this paper and will therefore not be explained in detail. Basically, classification within the stereo system is based on generating a mesh height model out of the point cloud obtained and applying adaptive thresholds to this mesh structure in order to characterize roadway, curb, and obstacles. The mono vision system is based on a approach similar to that of Thrun et al. (2006).

Prior to mapping the mono vision data into the grid data structure, the image must be transformed into the global world reference frame using the known camera calibration (Heikkilä & Silvén, 1996) and height information, which can easily be retrieved from the grid itself.

The creation of a sensor model for the 3D height data is more complex: First, a gradient field is calculated from the stored height profile. In Caroline's grid fusion system, the grid is mapped into image space by conversion into a grayscale image data structure, with intensity counting for cell height values. Subsequently, the Sobel operator is applied in both image directions. The results of both convolutions are summed and, after proper normalization, transformed back into the grid structure, storing the gradient values $\partial h/\partial x \partial y$ for each grid cell. Any existing obstacle will usually lead to a bigger peak within the gradient field, which can easily be detected. During the process of forward and reverse transformation, the grid structure in and out of a grayscale image would initially appear to be redundant, because the gradient operator could easily be applied to the height field itself. Yet by transforming the information into a commonly used image format, the broad variety of image processing algorithms and operators found in standard image processing toolkits, such as the OpenCV library (OpenCV Website, 2007), can easily be applied, thereby significantly reducing development time.

The acquired gradient values will then subsequently be mapped into a Dempster–Shafer representation, which leads to the desired sensor model combining all acquired height values. Similar to the method with the mono vision system, a mapping function is defined as follows:

$$
m_m(D) = \begin{cases} D_{\max}, & \left|\dfrac{\partial h}{\partial x \partial y}\right| \le G_{D_{\max}} \\ 0, & G_{D_{\max}} < \left|\dfrac{\partial h}{\partial x \partial y}\right| \le G_{U_{\min}}, \\ 0, & \left|\dfrac{\partial h}{\partial x \partial y}\right| > G_{U_{\min}} \end{cases}
$$

$$
\begin{aligned}
m_m(U) &= \begin{cases} 0, & \left|\dfrac{\partial h}{\partial x \partial y}\right| \le G_{D_{\max}} \\ \dfrac{U_{\max}}{G_{U_{\min}} - G_{D_{\max}}} \cdot \left(\left|\dfrac{\partial h}{\partial x \partial y}\right| - G_{D_{\max}} \right), & G_{D_{\max}} < \left|\dfrac{\partial h}{\partial x \partial y}\right| \le G_{U_{\min}}, \\ U_{\max}, & \left|\dfrac{\partial h}{\partial x \partial y}\right| > G_{U_{\min}} \end{cases} \\
m_m(N) &= 1 - m_m(D) - m_m(U),
\end{aligned} \tag{11}
$$

with $D_{\max}$ and $U_{\max}$ serving as parameters for maximum drivability/undrivability assigned to the gradient field, $G_{D_{\max}}$ being the maximum gradient value that is still considered to be fully drivable and $G_{U_{\min}}$ the minimum gradient value that is considered to be fully undrivable. By carefully tuning those four parameters, it is possible to suppress unwanted smaller gradients resulting, e.g., from unimportant depressions and knolls in the road while supporting higher gradients as originating from curbs or berms in order to correctly fuse this information into the grid cells by using Eq. (8).

4.2. Vision

Caroline's computer vision system consists of two separate systems. The first is a monocular color segmentation-based system that classifies the ground

in front of the car as drivable, undrivable, or unknown. It assists in situations in which the drivable terrain and the surrounding area (e.g., grass, concrete, or shrubs) differ in color. The output of this algorithm contributes to the grid-based fusion as described in Section 4.1.3. The second vision system is a multiview lane detection that identifies the different kinds of lanes described by DARPA, such as broken and continuous as well as white and yellow lane markings. Using four high-resolution color cameras and state-of-the-art graphics hardware, it detects its own lane and the two adjacent lanes to the left and right with a field of view of 175 deg at up to 35 m. The output of the lane detection algorithm is directly processed by the AI.

4.2.1. Lane Detection

Detecting lane markings on roads in an urban environment is a difficult but very important task. Although concepts exist that depend on additional markings, such as magnetic bands in the street, a more useful method must make intelligent use of what is available on today's roads. Toward this goal, we developed a lane detection system that is capable of analyzing several high-resolution images simultaneously and in real time. Our lane fitting algorithm uses a very versatile lane model and is robust with respect to outliers and artifacts. It also takes into account lane markings of adjacent lanes. It copes with different road setups, lane markings, and lighting situations. The lane detection process is divided into four parts, as shown in Figure 11. First, the raw images are downloaded from the cameras via the IEEE 1394b interface. Second, they are uploaded to graphics hardware, the color information is retrieved from the raw Bayer pattern, and the images are transformed into a single top-view perspective (Figure 12). Third, lane marking features are detected in the image (Figure 13). In the last step, a lane model is adjusted to match the features detected.

Data acquisition Three cameras with field of view of 58 deg cover the area in front of the car. A 22-deg-telephoto-lens camera provides a high-resolution view of the street ahead of the car. The four $1{,}376 \times 600$ 8-bit raw Bayer images are synchronously acquired via the IEEE 1394b interface at 14 frames per second (fps). The images are uploaded to the graphics card and converted to the red–green–blue (RGB) color space using bilinear interpolation. As the lane fitting algorithm works in a global coordinate system, the position and rotation of the vehicle, also referred to as ego state, must be available. A transformation function $f_{\text{ego}} : p_{\text{car}} \mapsto p_{\text{world}}$ can be defined if the ego state is known, where p_{car} is a point in the car's reference system and p_{world} is a point in a global Cartesian reference system. An inertial measurement unit (IMU) corrected by a GPS signal was used to generate the ego state.

Multiview fusion Because local changes in the light intensity are an indicator for white lines and local saturation changes indicate colored lane markings, the RGB images are converted to the HSV (hue, saturation, value) color space. This color space encodes saturation and color in separate channels. Knowing the intrinsic and extrinsic parameters of the camera, and including the orientation of the vehicle (pitch and roll), a lookup function that converts top-view coordinates to image coordinates can be used to create a single HSV top-view image. The lookup operation is applied to each source image. In regions where the projected images overlap, precedence $I_{\text{tele}} > I_{\text{middle}} > I_{\text{left}} > I_{\text{right}}$ is maintained as shown in Figure 12. The region

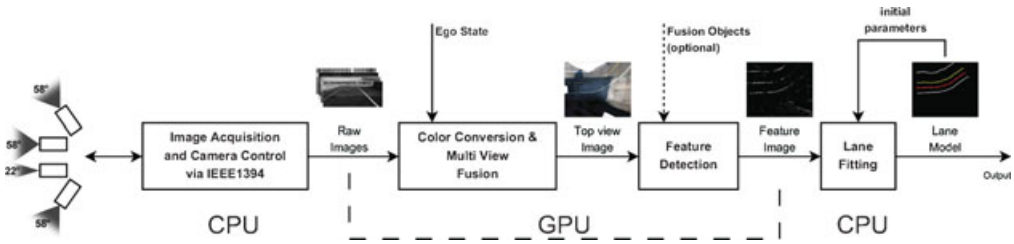
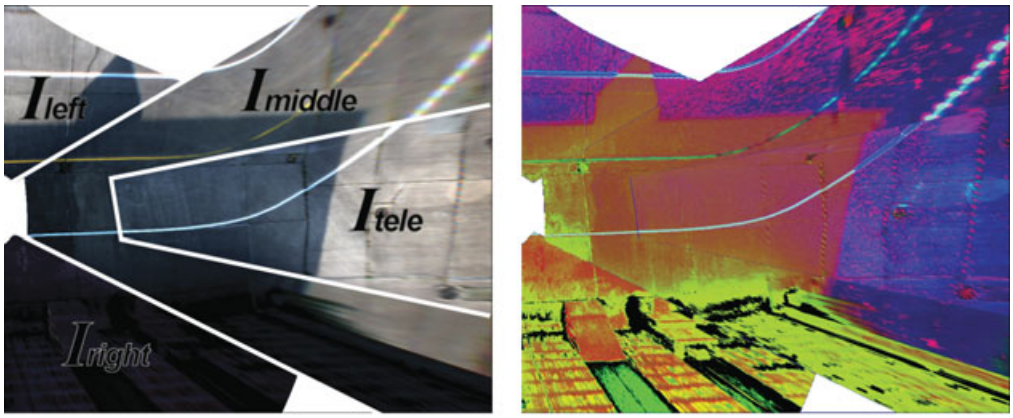
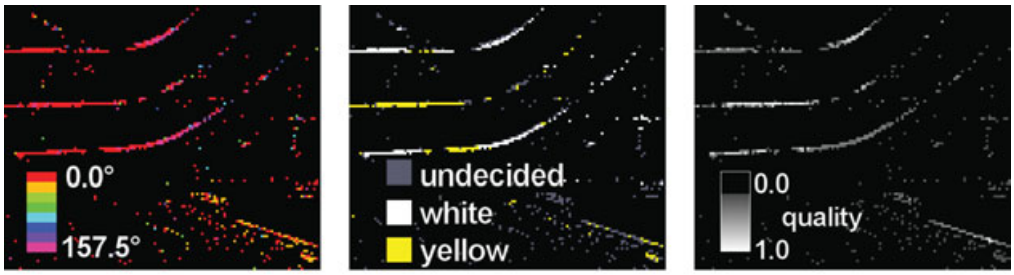


Figure 11. The four stages of the lane detection algorithm.



(a) RGB top view (b) HSV top view

Figure 12. The four different images (a, RGB color space used for visualization) are merged to a single HSV top-view image (b).



(a) Direction information (red channel) (b) Color information (green channel) (c) Quality information (blue channel)

Figure 13. The direction (a), color (b), and quality (c) of the features are encoded in an RGB image downloaded from the graphics card. For visualization purposes, the channels encoding the direction (a) and color (b) are colorized.

of interest covers the area of up to 30 m in front of the vehicle and 12 m to the left and right at a scale of 35 pixels per meter.

Features Lane markings can be described as a thin pattern of local differences of the road surface that cover long distances. Therefore, the basic concept underlying feature detection involves identification of these local differences in regions of 8×8 pixels that resemble road patches of approximately 25×25 cm. Analyzing the HSV top-view image, the feature detection's output is a downsampled feature image that encodes the quality, direction, and color, i.e., white or yellow, of the lane features in Figure 13. As lane markings exist in various colors, qualities, and widths and appear differently under changing lighting conditions, only a few stringent assumptions apply. When analyzing the top-view image for features, we check three criteria that must be present:

1. The local contrast v_{diff} must exceed a certain threshold. The local contrast is the difference between the local minimal and maximal values: $v_{\text{diff}} = v_{\max} - v_{\min}$.
2. Analyzing a local adaptive histogram, the distance b_{diff} between the two largest bins b_{high} and b_{low} must exceed a certain threshold. This is because it can be assumed that b_{low} contains pixels depicting the street and b_{high} identifies the lane marking.

3. The pixels in b_{high} must have a recognizable shape and orientation. For several discrete orientations, the ratio of the variances of the pixels' x and y coordinates is checked.

A detailed description is given in Algorithm 1 (see also Figure 14). As this algorithm is prone to discretization errors, supersampling improves the quality of the feature detection.

Algorithm 1 Feature detection algorithm

Data: An 8×8 region of a HSV top view image, thresholds t_{con}, t_{hist}, t_{dir} and t_{col}
Result: A feature quality q, direction $a \in \{ 0, 22.5, \ldots, 157.5 \}$ and color $c \in \{ \textit{white, yellow, undecided} \}$

for *the saturation and lightness channel* **do**
  $v_{diff} = v_{max} - v_{min}$; v_{max} and v_{min} are the maximal and minimal values of the current channel
  If $v_{diff} < t_{con}$ **then**
    break;
  end
  compute adaptive histogram;
  determine two largest bins b_{high} and b_{low}, Fig. 14(b) ;
  $b_{diff} = b_{high} - b_{low}$;
  If $b_{diff} < t_{hist}$ **then**
    break;
  end
  set of pixels p_{high} = pixels in b_{high};
  determine center of mass R of p_{high};
  initialize r_{max} and a_{max} to 0;
  for $i = 0; i <= 157.5; i = i + 22.5$ **do**
    rotate p_{high} around R by i degrees. determine ratio of variances $r = \frac{Var(X)}{Var(Y)}$;
  end
  if $r_{max} < t_{dir}$ then
    break;
  end
  label this region as a feature;
  If *current channel is lightness* **then**
    $q_{white} = b_{diff}$; $a_{white} = a_{max}$
  else
    $q_{yellow} = b_{diff}$; $a_{yellow} = a_{max}$
  end
end
If $q_{white} > t_{col}$ & $q_{white} > q_{yellow}$ **then**
  $c = white$; $a = a_{white}$
end
if $q_{yellow} > t_{col}$ & $q_{yellow} > q_{white}$ **then**
  $c = yellow$; $a = a_{yellow}$
end
$q = max(q_{white}, q_{yellow})$;

Lane model The lane model consists of connected lane segments. Each segment s_i is described by a length l_i (given parameter), a width w_i and an angle $d_i = \alpha_i - \alpha_{i-1}$ describing the difference of orientation between this segment and the previous one as shown in Figure 15. The first segment is initially placed on the current coordinates of the vehicle and facing in the driving direction, assuming that the vehicle is actually located on the street. Knowing the position c_0 of the initial segment as well as the lengths l_i and the angular changes d_i of all segments, the position p_i and global orientation α_i of each segment can be computed. Each segment contains information as to whether the vehicle's lane is confined by lane markings and whether additional lanes to the left and right exist. Straight streets, sharp curves, and a mixture of both can all be described by the model.

Lane fitting The main goal of the lane fitting algorithm is to find a parameter set for a lane model that explains the features found in the current top-view image and the previous frames. To create a global model of the lane, all feature points are mapped to world space coordinates and inserted into a list l_p. This is done using the function $f_{\text{ego}} : p_{\text{car}} \mapsto p_{\text{world}}$ defined by the current ego state. Old data, i.e., feature points gathered during previous frames, may be kept if the features of a single image are too sparse. For each frame, the existing lane model or an initial guess is used to define four regions of interest as shown in Figure 16(a). These are the regions expected to contain the own lane's markings and the lane markings of the adjoining lanes. If a feature is inside such a region, it is labeled as *outer left*, *left*, *right*, or *outer right*. Otherwise, it is discarded. Afterward, features from previous frames are mixed with the new data as depicted in Figure 16(b).

As the first currently visible segment s_f of the lane model is determined, older segments are no longer considered. If the list of lane segments is empty, it is initialized with $s_0 \leftarrow s_f$. Starting from s_f, each segment s_i is estimated (or reestimated if it has previously been estimated). Therefore, an initial guess as to the orientation α_i of s_i is made as shown in Figure 15. All local features relevant for estimating s_i are rotated by α_i around the starting point p_i of s_i. A RANSAC algorithm is used to estimate the parameter d_i and w_i: Iteratively, two feature points p_x and p_y are chosen. Assuming that they are located on the lane markings they were labeled for, the gradient $g_i = m_i / l_i$ as well as the width w_i are derived from their coordinates. All features that are also

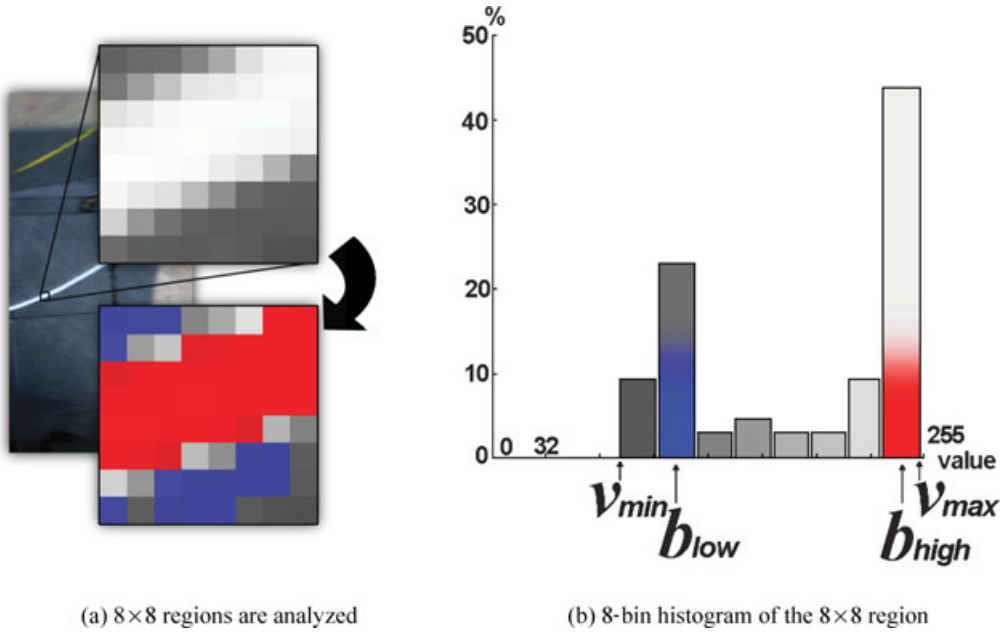


Figure 14. 8×8 pixel regions of the top-view image (a, top) are tested for possible features. The distance between the two largest bins b_{low} (b, blue) and b_{high} (b, red) of the histogram determines the quality of the feature. The pixels gathered in b_{high} must be arranged in a directed shape (a, red area).

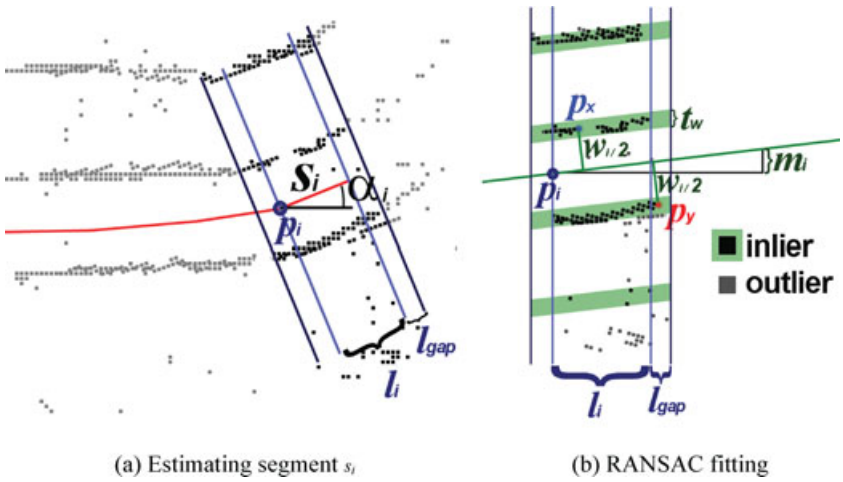


Figure 15. p_i, α_i, l_i, and l_{gap} identify the features relevant for s_i. After rotating around α_i, a RANSAC fitting eliminates outliers among the features.

sufficiently described by g_i and w_i are counted as inliers. This process is repeated n times, and the parameter set with the most inliers is used to define s_i. A quality function q takes into account the ratio of inliers and outliers, the amount of inliers, and the quality of the features and states the quality of the segment. The quality is computed for every region of interest (outer left, left, right, and outer right). If the maximum of these qualities exceeds a threshold t_q, the segment is considered to be valid and the next

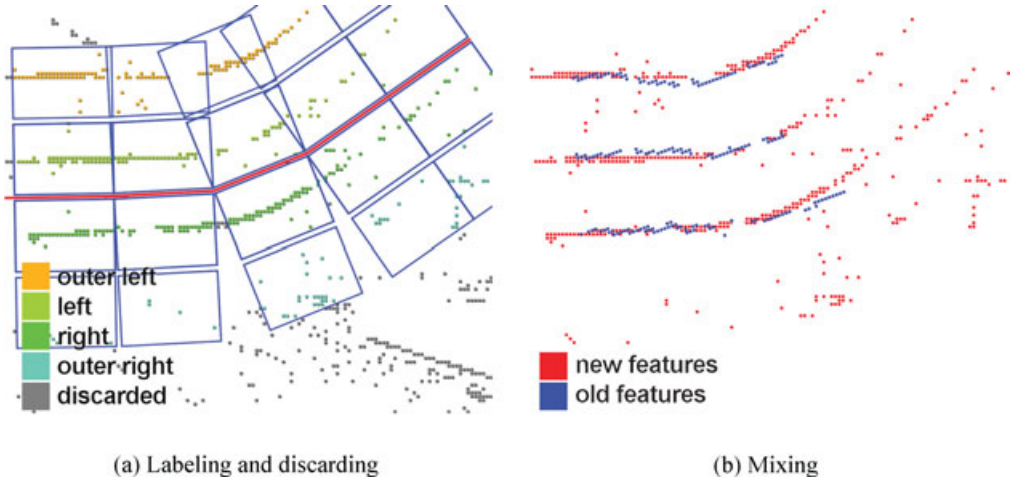


Figure 16. Regions of interest (a, blue boxes) determine to which lane marking features are assigned. Afterward, old and new features are mixed (b).

segment s_{i+1} is estimated. After all segments are estimated as shown in Figure 17, a proposal about the lane markings' colors can be made by looking at the inliers' average color.

Results and evaluation The algorithm was thoroughly tested on several sites in northern Germany and Texas. A frame rate of 10 fps could be maintained using a 2-GHz Intel Core 2 Duo with a GeForce 7600 GTS graphics card. The testing sessions included different weather and lighting conditions. The amount of false positives was reduced significantly by utilizing the vehicle's other sensors. The objects detected by LIDAR and radar sensors were used to mask out regions in the feature image where other cars, walls, cones, and poles caused irritating artifacts in the top-view image.

4.2.2. Area Processor

The area processor consists of a single color camera whose images are interpreted by a color segmentation algorithm suitable for urban environments. This algorithm separates an image into areas of drivable and nondrivable terrain. Assuming that a part of the image is known to be drivable terrain, other parts of the image are classified by comparing the Euclidean distance of each pixel's color to the mean colors of the drivable area in real time. Moving the search

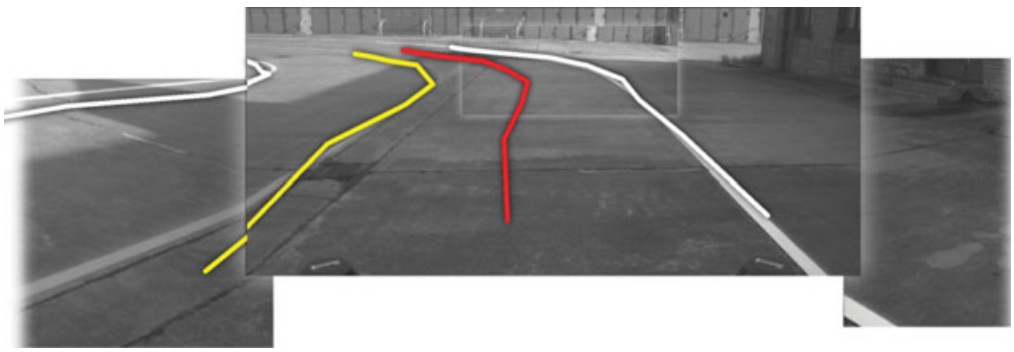


Figure 17. The lane model reprojected onto the original images.

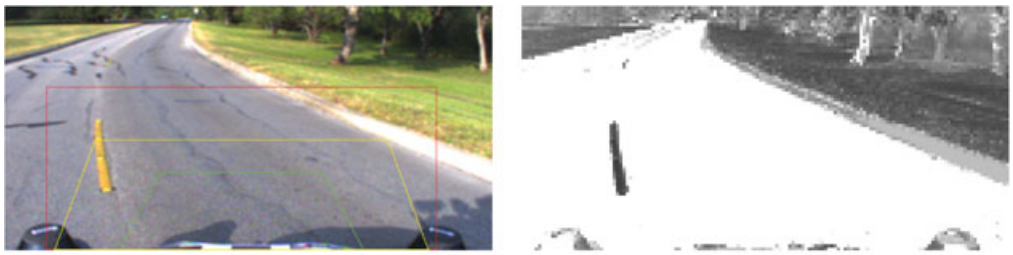
area depending on each frame's result ensures temporal consistency and coherence. Furthermore, the algorithm classifies artifacts such as white and yellow lane markings and hard shadows as areas of unknown drivability. Although Caroline is able to perform basic driving tasks without this algorithm, it is needed in situations when terrain cannot be distinguished by other sensors, i.e., sections without proper lane markings, streets without high curbs, and off-road tracks.

Related work As a foundation for the area detection algorithm, we used the real-time approach suggested by Thrun et al. (2006) in the 2005 DARPA Grand Challenge. The basic idea is to consider a given region in the actual image as drivable. The predominant mean color values in that area are retrieved and compared to the pixel values in the entire image. Similar pixels are marked as drivable. The algorithm was designed for off-road terrain; therefore it cannot be applied to urban scenarios without fundamental modifications. We will describe the algorithm in the next section. The expectation maximization (EM) algorithm used for color clustering in this approach is thoroughly described in Duda and Hart (1973) and Blimes (1997). Instead of the EM algorithm, the KMEANS algorithm that we used during the competition is also suitable for color clustering, as described in Bradski, Kaehler, & Pisarevsky (2005). An algorithm similar to the one mentioned above points out the advantage of color spaces other than RGB (Ulrich & Nourbakhsh, 2002).

The Stanford University algorithm for detecting drivable terrain The main idea of the algorithm is to use the output of the laser scanner, normally a scan line, which is integrated over time to a height map in world coordinates. A polygon is defined that covers an area in front of the car identified as level and therefore as a drivable surface. This polygon is transformed into image coordinates from the camera and clipped to the image boundaries. The resulting polygon is considered as the area that is drivable. In this area the pixels' color values are collected and clustered by color, for example, bright gray and yellow. These color clusters are compared to the color values of each pixel in the image using distance measurements in the color space. If a resulting distance is smaller than a given threshold, the area comprised by the pixel is marked as drivable. The main benefit of the algorithm is that the range in which drivability can be estimated is enhanced from only a few meters to more than 50 m.

Problems arising in urban and suburban terrain Designed for competing in a 60-mile (96.5 km) desert course, the basic algorithm succeeds well in explicit off-road areas, which are limited by sand hills or shrubs. When tested in urban areas, new problems occur because there are streets with lane markings in different colors or tall houses casting long shadows. The yellow lane markings are often not inside the area of the polygon P_{scanner} (output of the laser scanner), so they are not detected as drivable. In particular, nondashed lines prohibit a lane shift as shown in Figure 18, and stop lines seem to block the road.

Another problem is shadows cast by tall buildings during the afternoon. Small shadows from trees in a fairly diffuse light change the color of the street only slightly and can be adapted easily. But huge and dark shadows appear as a big undrivable area as shown in Figure 19. Even worse, once inside a shadowed area, the camera auto exposure



(a) Original picture (b) Normal drivability grid

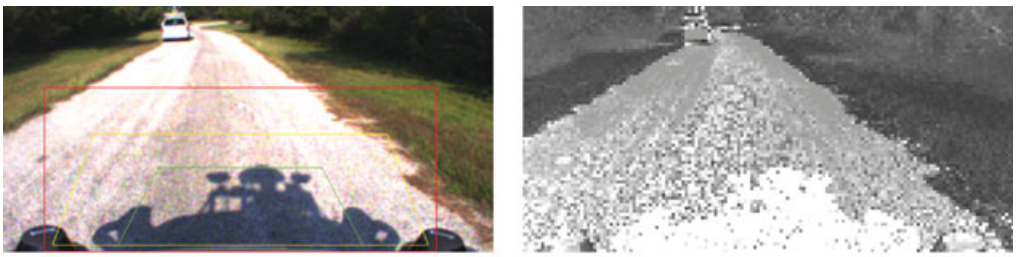
Figure 18. The drivability grid (b) depicts the output of algorithm; the results differ from black (undrivable) to white (drivable). A yellow line (a) is marked as undrivable (b, black) because the color differs by too much from the street color.

(a) Original picture (b) Normal drivability grid

Figure 19. Large, dark shadows (a, left) differ too much from the street color (b, dark).

(a) Original picture (b) Normal drivability grid

Figure 20. Exposure is automatically adapted inside shadows (a). Areas outside the shadow are overexposed and are marked as undrivable (b, dark).



(a) Original picture (b) Normal drivability grid

Figure 21. The vehicle's own shadow can lead to problems (a), for example, if only the shadowed region is used to detect drivable regions (b, white).

adapts to the new light situation, such that the area outside the shadow becomes overexposed and appears again as a big undrivable area as depicted in Figure 20.

Another problem during the afternoon is the car's own shadow, in this paper referenced as "egoShadow," when the sun is behind the car. Sometimes it is marked as undrivable; sometimes it is completely adapted and marked as drivable, but the rest of the street is marked as undrivable as shown in Figure 21. A fourth problem occurs when testing on streets without curbs but limited by mowed grassy areas. The laser scanner does not recognize the grass as undrivable, because its level is about the same as the street niveau. This causes the vehicle to move onto the grass, so that colors are adapted by the area processing algorithm and consequentially keeps the car on the green terrain.

Alterations to the basic algorithm Differing from the original algorithm, our implementation does not classify regions of the image as drivable and undrivable. The result of our distance function is mapped to an integer number ranging from 0 to 127, instead of creating a binary information via a threshold. In addition, a classification into the categories "known drivability" and "unknown drivability" is applied to each pixel. These alterations are required because the decision about the drivability of a certain region is not made by the algorithm itself but by a separate sensor fusion application. For the sake of performance the KMEANS nearest-neighbors algorithm was chosen instead of the EM algorithm, because the resulting grids are of almost the same quality but the computation is considerably faster. Tests have shown that better results can be achieved by using a color space that separates the luminance and the chrominance in different channels, e.g., HSV, LAB (lightness, a, b = color opponent coordinates), YUV (luminance, chrominance (u, v)). The problem with HLS (hue, lightness, saturation) and HSV is that chrominance information is coded in one hue channel and the color distance is radial. For example, the color at 358 deg is very similar to the one at 2 deg, but they are numerically very far away from each other. Thus a color space is chosen where chrominance information is coded in two channels, for example, in YUV or LAB, where similarity between two colors can be expressed as a Euclidean distance.

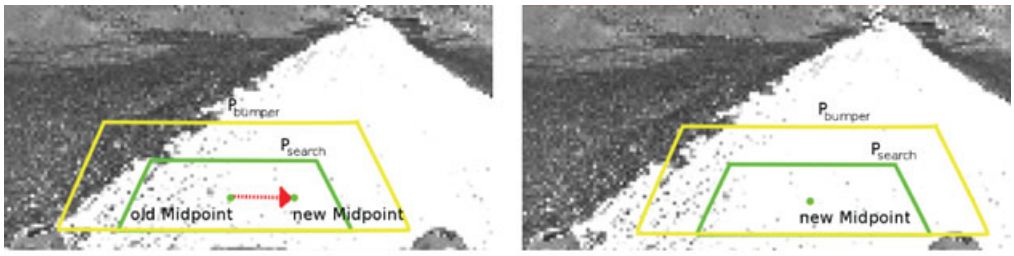
Preprocessing To cope with the problems of large shadows and lane markings, a preprocessing system was developed. Before the camera picture is processed, it is handed over to the following preprocessors: white preprocessor (masking out lane markings and overexposed pixels), black preprocessor (masking out large, dark shadows), yellow preprocessor (masking out lane markings), and egoShadow preprocessor (masking out the car's shadow in the picture). The output of each preprocessor is a bit mask (1: feature detected, 0: feature not detected), which is used afterward in the pixel-classifying process, to mark the particular pixel as "unknown," which means that the vision-based area processor cannot provide valid information about the area represented by that pixel. In the following, the concept of each preprocessor is described briefly:

White preprocessor To deal with overexposed image areas during shadow traversing, pixels whose brightness value is larger than a given threshold are detected. The preprocessor converts the given image into a HSV color space and compares the intensity value for each pixel with a given threshold. If the value is above the threshold, the pixel of the output mask is set to 1.

Black preprocessor As huge dark shadows differ too much from the street color and would therefore be labeled as impassable terrain, pixels whose brightness value is smaller than a given threshold are masked out. The preprocessor analogously converts the given image into a HSV color space and compares the intensity value for each pixel with a given threshold. If the value is below the threshold, the pixel of the output mask is set to 1.

Yellow preprocessor Small areas of the image that are close to yellow in the RGB color space are detected so that yellow lane markings are not labeled as undrivable but rather as areas of unknown drivability. For each pixel of the given image, the RGB ratios are checked to detect yellow lane markings. If the green value is larger than the blue value and larger or a slightly smaller than the red value, the pixel is not considered yellow. If the red value is larger than the sum of the blue and the green values, the pixel is also not considered yellow. Otherwise, the pixel is set to $\min(R, G)/B - 1$. Afterward, a duplicate of the computed bit mask is smoothed using the mean filter, dilated, and subtracted from the bit mask to eliminate huge areas. For different areas of the image, different kernel sizes must be applied. In the end, only the relatively small yellow areas remain. A threshold determines the resulting bit mask of this preprocessor

EgoShadow preprocessor When the sun is behind the car, the vehicle's own shadow appears in the picture and either is marked as undrivable or is the only area marked as drivable. Therefore, a connected area directly in front of the car is identified whose brightness value is low. At the beginning of the whole computing process a set of base points $p(x, y)$ is specified, which mark the border between the engine hood and the ground in the picture. The region of interest in each given picture is set to $y_{\max}$, the maximum row of the base points, so that the engine hood is cut off. From these base points the preprocessor starts a flood fill in a copy of each given image, taking advantage of the fact that the car's shadow appears in similar colors. Then the given picture is converted to HSV color space, and the flood-filled pixels are checked to determine whether their intensity value is small enough. Finally, the sum of the flood-filled pixels is compared to a threshold, which marks the maximum pixel area that constitutes the car's own shadow.



(a) Dynamic polygon at time t (b) Dynamic polygon at time $t+1$

Figure 22. How the dynamic search polygon (a, green trapezoid) is transposed to the right (b) because the calculated moment is positive in the x direction.

The dynamic search polygon Using the output of the laser scanner to determine the input polygon works quite well if the drivable terrain is limited by tall objects such as sand hills or shrubs. In urban terrain, however, the output of the laser scanner must be sensitized to level distances smaller than curbs (10–20 cm), which becomes problematic if the street moves along a hill where the distance is much greater. Thus, the laser scanner polygon does not remain a reliable source, especially because both modules solve different problems: The laser scanner focuses on range-based obstacle-detection (Ulrich & Nourbakhsh, 2000), which is based on analysis of the geometry of the surroundings, whereas the vision-based area processor follows an appearance-based approach. For example, driving through the green grass next to the street is physically possible and therefore not prohibited by a range-based detection approach, but it must be prevented by the appearance-based system. This led to the concept of implementing a self-dynamic search polygon (see Figure 22) that has a static shape but is able to move along both the X and the Y axes in a given boundary polygon P_{boundary}. The initial direction is zero. Every movement is computed using the output of the last frame's pixel classification. For the computation a bumper polygon P_{bumper} is added, which surrounds the search polygon. The algorithm proceeds in the following steps:

Algorithm 2 Dynamic search polygon algorithm

Data last frame's grid of classified pixels, actual bumper polygon P_{bumper}
Result: updated position of the Polygons P_{bumper}

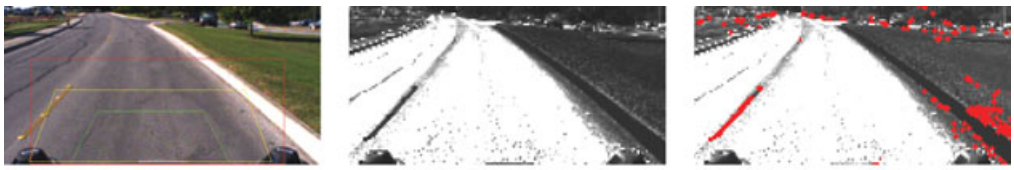
1 begin
2 Initialize three variables $pixelSum$, $weightedPixelSumX$, $weightedPixelSumY$ to zero
3 **foreach** *pixel of the grid which is inside the bumper* **do**
4 count the amount $pixelSum$ of visited pixels
5 *if drivability of the actual pixel is above a given threshold* **then**
6 Add the pixel's x-Position relative to the midpoint of P_{bumper} to $weightedPixelSumX$
7 Add the pixel's y-Position relative to the midpoint of P_{bumper} to $weightedPixelSumY$
8 **end**
9 **end**
10 Perform the division $x_{moment} = \frac{weightedPixelSumX}{pixelSum}$ and $y_{moment} = \frac{weightedPixelSumY}{pixelSum}$ and round the results to natural numbers
/* **The value** x_{moment} gives the amount and direction of the movement of P_{bumper} in x**-direction, the value** y_{moment} gives the amount and direction of the movement of P_{bumper} in y-direction.*/
11 Add the values x_{moment} and y_{moment} to the values of the actual midpoint of P_{bumper} to retrieve the new midpoint of P_{bumper}
12 Check the values of the new midpoint of P_{bumper} against the edges of $P_{boundary}$ and adjust the values if necessary
13 Add the values x_{moment} and y_{moment} to the values of the actual midpoint of the search polygon to retrieve the new midpoint of the search polygon as shown in Fig. 22
14 To prevent that the search polygon gets stuck in a certain corner, it is checked, if $x_{moment} = 0$ or if $y_{moment} = 0$
/* **For example** if $x_{moment} = 0$, **it is evaluated, if the midpoint of** P_{bumper} **is located right or left to the midpoint of** $P_{boundary}$; x_{moment} **is set to** 1, **if** P_{bumper} **is located left, otherwise it is set to** -1**. An analogous check can be performed for the** y_{moment}.*/
15 end

(a) Original picture (b) Normal drivability grid (c) With black preprocessor

Figure 23. The results with the black preprocessor. The picture in the center shows the classification results without the black processor. In the picture on the far right the critical region is classified as unknown (red).

(a) Original picture (b) Normal drivability grid (c) With white preprocessor

Figure 24. The results with the white preprocessor. The picture in the center shows the classification results without the white processor. In the picture on the far right the critical region is classified as unknown (red).



(a) Original picture (b) Normal drivability grid (c) With yellow preprocessor

Figure 25. The results with the yellow preprocessor. The picture in the center shows the classification results without the yellow preprocessor. In the picture on the far right the lane marks are classified as unknown (red).

Implementation and performance The algorithm was implemented with the Intel OpenCV library (OpenCV Website, 2007). The framework software is installed on an Intel Core 2 Duo Car PC with a Linux operating system and communicates with an IDS uEye camera via USB. The resolution of a frame is 640×480, but the algorithm applied downsampled images of size 160×120 to attain a manually adjusted average performance of 10 fps. The algorithm is confined to a region of interest of 160×75 cutting of the sky and the engine hood.

In Figure 23 the difference between normal area processing and processing with the black preprocessor is shown. Without the preprocessor, the large shadow of a building to the left of the street is too dark to be similar to the street color and is classified as undrivable. The black preprocessor detects the shadowy pixels, which are classified as unknown (red).

The problem of overexposed areas in the picture is shown in Figure 24, where the street's color outside the shadow is almost white and therefore classified as undrivable in the normal process. The white preprocessor succeeds in marking the critical area as unknown, so that the vehicle has no problem in leaving the shadowy area.

Yellow lane markings differ from pavement in color space so that a human driver can easily detect them even under adverse lighting conditions. This advantage turns out to be a disadvantage for a standard classification system, which also classifies the lane markings as undrivable, as shown in Figure 25:

Lane markings are interpreted as tiny walls on the street. To counteract this problem, we use a preprocessing step that segments colors similar to yellow. To deal with different light conditions, the color spectrum must be wider so that a brownish or grayish yellow is also detected. This leads to some false positives as shown in Figure 25, but the disturbing lane markings are clearly classified as unknown. The vehicle is now able to change lanes without further problems.

A problem with the vehicle's own shadow occurs only when the sun is located behind the vehicle, but in these situations the classification can deliver insufficient results. Figure 26 shows the shadowy area in front of the car as unknown.

The benefit of a search polygon that is transposed by the output of the last frame is tested by swerving about so that the car moves very close to the edges of the street. Figure 27 shows the results when moving the car close to the left edge. As the static polygon touches a small green area, a somewhat green mean value is gathered and so the resulting grid shows a certain amount of drivability in the grassland, whereas the dynamic polygon moves to the right of the picture to avoid touching the green pixels so that the resulting grid does not show drivability on the grassland.

4.3. Artificial Intelligence

4.3.1. The DAMN Architecture

To control Caroline's movement, the artificial intelligence computes a speed and a turning wheel angle for every discrete step. Turning the steering wheel results in different circle radii on which the car will move. Instead of the radii, the approach is based on the inverse, a curvature.

A curvature of 0 represents driving straight ahead, whereas negative curvatures result in left and positive curvatures in right turns as shown in Figure 28.

This curvature, as the most important factor to influence, is selected in an arbiter as described in the DAMN architecture (Rosenblatt, 1997). This architecture models each input as behavior, which gives a vote for each possible curvature. More behaviors can be added easily to the system, which makes it very modular and extendable. The following behaviors are considered:

- Follow way points: Simply move the vehicle from point to point as found in the RNDF.
- Stay in lane: Vote for a curvature that keeps Caroline within the detected lane markings.

Figure 26. The results with the egoShadow preprocessor. The picture in the center shows the classification results without the egoShadow processor. In the picture on the far right the car's own shadow is classified as unknown (red).

Figure 27. The same frame first computed with a static search polygon (a, b), and then with the dynamic polygon (c, d). The dynamic movement calculation caused the polygon to move to the right (c).

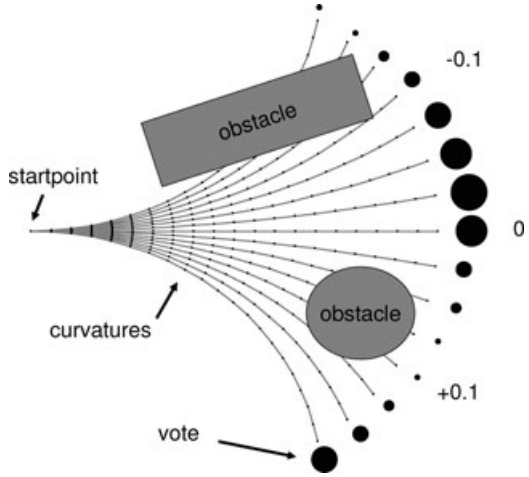


Figure 28. Curvature field: Larger black circles represent preferred votes.

- Avoid obstacles: Vote for curvatures that keep the vehicle as far away from obstacles as possible and forbid curvatures leading directly into them.
- Stay on roadway: Avoid curb-like obstacles detected by grid-based fusion with laser scanners and color camera.
- Stay in zone: Keep the vehicle in the zone, defined by perimeter points in the RNDF.

All collected votes are weighted to produce an overall vote. The weights again are not fixed; they depend on factors including distance to an intersection, presence of lanes, and more. A trajectory point is calculated by following the best-voted curvature for 1 m. A trajectory point holds information such as position, orientation, and speed. Starting at this trajectory point, all behaviors vote again for curvatures to find the next point until a list of points is computed. This list has to be long enough to come to a complete stop at current speed. The speed is controlled by another arbiter influenced by different behaviors, which each provides a maximum speed. The arbiter simply selects the lowest of these speeds. These behaviors are RNDFMax, sensor health, zone, reverse, safety zone, obstacle distance, and following other obstacles. On the basis of the trajectory points calculated iteratively, we design a drivable corridor for further processing by the next module in the chain, the path planner.

4.3.2. Interrupts

Because the AI has to deal with more complex situations, e.g., stopping at a stopline and yielding the right-of-way, than the DAMN architecture is designed for, we extended DAMN by an interrupt system. At each trajectory point found, each interrupt is called upon to decide whether it wants to be activated at its location. If so, the speed stored in the trajectory points is reduced to bring the car to a smooth stop. If the point is reached, the interrupt is activated and the arbiters are stopped until the interrupt returns control to the arbiters. Some of our interrupts are as follows:

- Intersection: Activated at a stopline until it is our turn.
- Queue: Wait in a line at an intersection.
- Overtake: Stop the car when the lane is blocked and wait for other lane to clear to start passing maneuver.
- U-turn: Activated at a dead-end street; this interrupt actually performs the U-turn and turns the car around.
- Road blocked: Activated if the entire road is blocked. This interrupt then activates the U-turn interrupt when appropriate.
- Parking: Activated at a good alignment in front of the parkbox; this interrupt returns control after the parking maneuver is finished.
- Pause: Active as long as the car is in pause mode.
- Mission complete: Final checkpoint is reached.

An example can be seen in Figure 29, where the queueing interrupt has to be activated at some point in the future and the speed must therefore be reduced.

4.3.3. Example

An example of how different behaviors interact is shown in Figure 30. In the recorded situation, Caroline just started overtaking another car, blocking its lane. The plots represent the calculation of one trajectory: 20 trajectory points are calculated from the front to the back. For each point votes for 40 curvatures are made; these are displayed from left to right.

The lane behavior [Figure 30(a)] demands a sharp left for the first four curvatures and then a right

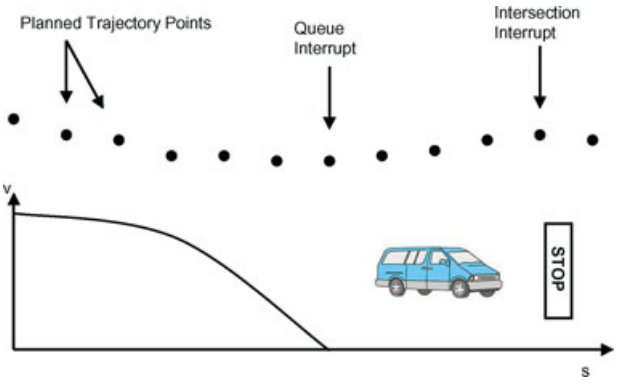


Figure 29. Interrupt example.

turn, which finally transitions to straight driving. This would bring Caroline quickly to the free lane to pass the obstacle vehicle. The obstacle behavior [Figure 30(b)] has two obstacles affecting the votes: On the left, a wall forbids going farther to the left; on the right one can see the car that is be passed. Finally the way point behavior [Figure 30(c)] wants to go to the right all the time, because that is the lane where Caroline should be and where the way points are, but is outvoted by the other behaviors in Figure 30(d).

4.4. Vehicle Control

Lateral and longitudinal controls are the basics of autonomous vehicle guidance. In the following, both concepts as installed in Caroline for the DARPA Urban Challenge are discussed in detail.

4.4.1. Longitudinal Control

Whereas the maximum and minimum speeds of the vehicle are chosen by the AI, the controller must calculate the braking and accelerator set points in order to maintain a given speed.

For this purpose, the longitudinal controller is separated into outer- and inner-loop controllers. Based on the given speed set point, the outer-loop

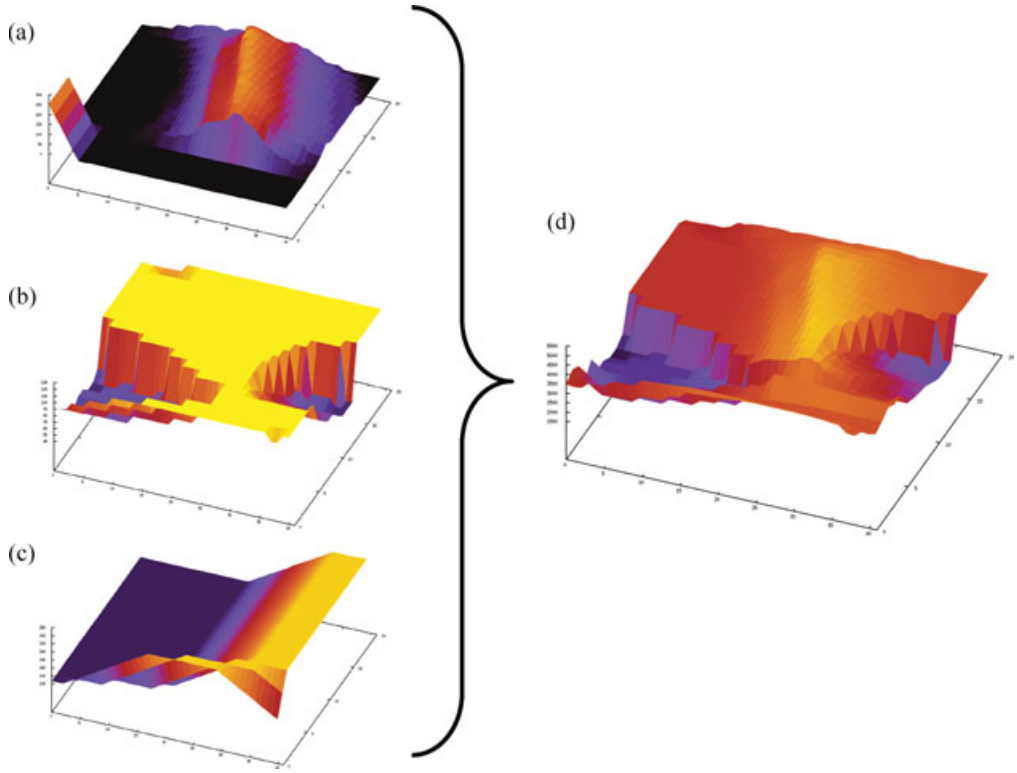


Figure 30. Votes of (a) stay in lane, (b) avoid obstacles, (c) follow way points, and (d) weighted sum.

controller determines the required acceleration. Finally, the inner-loop controller calculates throttle and brake input to track the required acceleration. The acceleration of the vehicle, which is needed for feedback of the lower controller, is provided in high resolution by the GPS/INS system.

Gear shifting is handled via an automatic gear box. However, to switch between forward, backward, and parking states, an automatic lever arm is attached at the gearshift. The lever arm position can be commanded with a CAN interface.

Longitudinal dynamics The driving power must be greater than the sum of all driving resistances, that is, the sum of rolling, air, and acceleration resistance. Engine torque M_M is a function of throttle α_A, engine speed n_M, and engine acceleration $\dot{n}_M$:

$$M_M(\alpha_A, \dot{n}_M, \dot{n}_M) = \frac{r}{\eta_k\, i_k}\left[f_R\, m g + c_w\, A\, \frac{\rho}{2}\left(\frac{n_M\, 2\pi\, R_0}{i_k}\right)^2 + \lambda\, m \frac{\dot{n}_M\, 2\pi\, R_0}{i_k}\right]. \quad (12)$$

The meanings of the parameters are given in Table I.

The model is used for the inner-loop controller to simulate different control strategies for the longitudinal control. The plant model for the outer-loop controller is the transfer function between desired vehicle acceleration and actual vehicle speed. The inner loop is approximated as a PT1 element. In addition, an integral element is needed to integrate the speed from acceleration:

$$P(s) = \frac{1}{s\,(T\,s+1)}. \quad (13)$$

Table I. Longitudinal model parameters.

Symbol	Parameter
R_0	Wheel radius, unloaded
r	Wheel radius, loaded
η_k	Degree of efficiency, gear box
i_k	Gear transmission ratio
f_R	Rolling friction factor
m	Mass
g	Gravity
c_w	Air resistance factor
A	Cross-sectional area
ρ	Air density
λ	Molding bodies factor

Introducing measured values of the drive chain into the model leads to a value of $T = 0.6$ s for system lag.

P-PD-control controller cascade As mentioned above, the longitudinal controller is separated into outer and inner control loops. The block diagram in Figure 31 depicts the control structure. $K(s)$ stands for each transfer function of the different controller parts. Different control parameters are used for acceleration and deceleration. Whereas a PD controller is applied for the inner loop, a P controller is introduced for the outer control loop. Control outputs for acceleration and braking are combined via a predefined logic to prevent the system from activating throttle and brake at the same time.

In addition, an engine map can be used for direct feed forward of the throttle. Figure 32 shows a typical implementation of an engine map for longitudinal control.

Performance of the longitudinal controller Figure 33 illustrates the performance of the

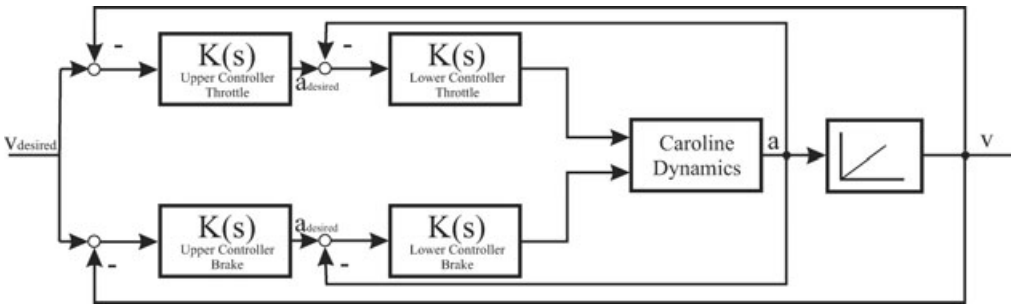


Figure 31. Block diagram of the longitudinal controller.

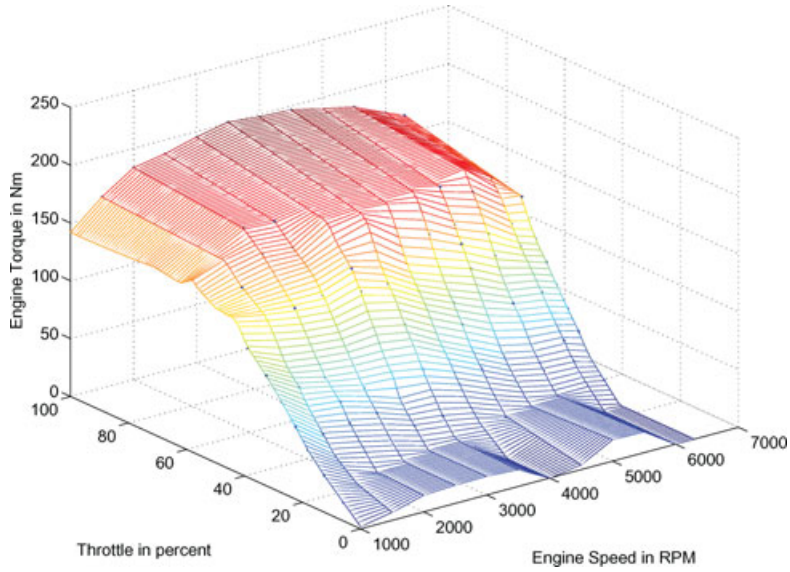


Figure 32. Engine map.

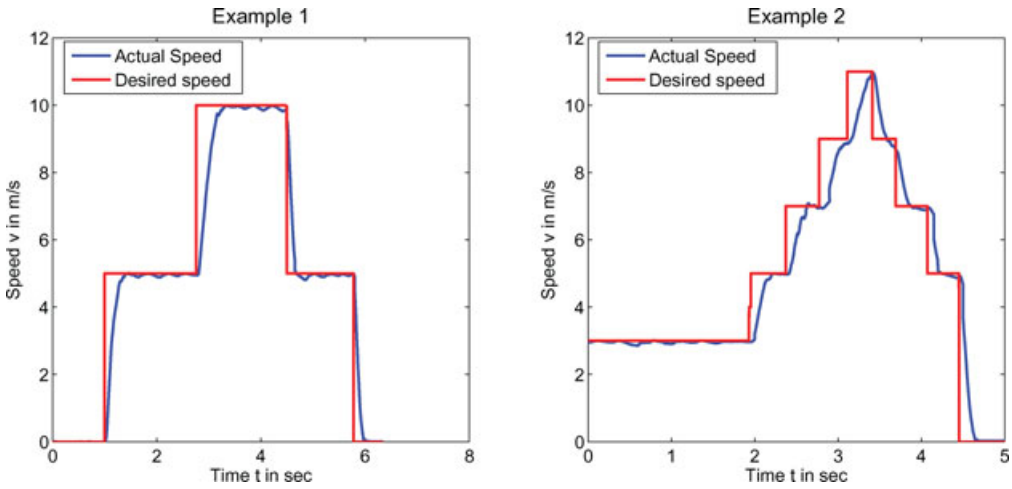


Figure 33. Performance of the longitudinal controller.

longitudinal control strategy. Two different examples are shown with two different speed profiles. Whereas in the first example, the desired speed is changed in long and large steps, in the second example the speed is changed in shorter and smaller steps. The desired as well as the actual speeds of Caroline are illustrated.

4.4.2. Lateral Control

It is the main goal of the lateral controller to follow a given trajectory with a minimum of track error. Second, vehicle driving maneuvers should match certain comfort parameters for a smooth driving experience.

Vehicle dynamics For simulation of the vehicle as well as design of the controllers, it is

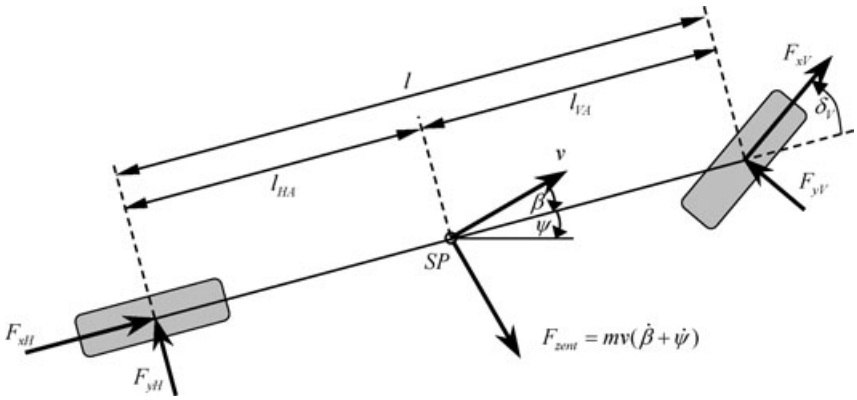


Figure 34. Bicycle model.

necessary to describe motion behavior with a mathematical model. In the following the bicycle model is used (Figure 34). The bicycle model is based on the following assumptions:

- The center of mass of the car is located at street level.
- Two wheels of each axle are combined as one wheel in the center of the axles.
- The longitudinal acceleration is zero.
- The wheel load of all wheels is constant.
- Lateral forces at the wheel are proportional to skew angle.

A state space representation within the following structure is preferred:

$$\dot{\mathbf{x}}(t) = \underline{\underline{A}}\,\mathbf{x}(t) + \underline{\underline{B}}\,\mathbf{u}(t) + \underline{\underline{E}}\mathbf{z}(t), \quad \mathbf{x}(0) = \mathbf{x}_0. \tag{14}$$

Track error and track angle deviation have to be described mathematically to take them into consideration. Track angle deviation is defined as the difference between desired and actual orientation of the car. It is assumed that the derivation of the track angle ζ_{desired} can be calculated as the product of the curvature κ of the track and the current speed v:

$$\zeta_{\text{desired}} = \kappa \cdot v. \tag{15}$$

Yaw angle ψ_{rel} with respect to the desired track is the difference between absolute yaw angle ψ and desired track angle ζ_{desired}:

$$\psi_{\text{rel}} = \psi - \zeta_{\text{desired}}. \tag{16}$$

As a result, yaw rate $\dot{\psi}_{\text{rel}}$ with respect to the desired track can be determined:

$$\dot{\psi}_{\text{rel}} = \dot{\psi} - \kappa\, v. \tag{17}$$

Moreover, the derivation of the track error $\dot{d}$ can be formulated based on speed v, attitude angle β, and relative yaw angle ψ_{rel}:

$$\dot{d} = v\,(\beta + \psi_{\text{rel}}) \tag{18}$$

The state space representation of the bicycle model can be combined with the mathematical representation of the track error, track angle deviation, and an additional time delay T_L between commanded and actual steering wheel angles. The state vector consists of yaw rate $\dot{\psi}$, attitude angle β, relative yaw angle ψ_{rel}, track error d, and actual steering angle δ. The result is the following state space model with the commanded steering angel δ_{desired} as the input variable and curvature κ as the outer noise:

$$\begin{pmatrix} \ddot{\psi} \\ \dot{\beta} \\ \dot{\psi}_{\text{rel}} \\ \dot{d} \\ \dot{\delta} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 & a_{15} \\ a_{21} & a_{22} & 0 & 0 & a_{25} \\ 1 & 0 & 0 & 0 & 0 \\ 0 & v & v & 0 & 0 \\ 0 & 0 & 0 & 0 & -1/T_L \end{pmatrix} \cdot \begin{pmatrix} \dot{\psi} \\ \beta \\ \psi_{\text{rel}} \\ d \\ \delta \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ i_L/T_L \end{pmatrix} \cdot \delta_{\text{desired}} + \begin{pmatrix} 0 \\ 0 \\ -v \\ 0 \\ 0 \end{pmatrix} \cdot \kappa \tag{19}$$

with

$$a_{11} = -\frac{c_{\alpha V}\, l_V^2 + c_{\alpha H}\, l_H^2}{\theta\, v}, \quad a_{12} = -\frac{c_{\alpha V}\, l_V + c_{\alpha H}\, l_H}{\theta}, \quad a_{15} = \frac{c_{\alpha V}\, l_V}{\theta}, \tag{20}$$

$$a_{21} = -1 - \frac{c_{\alpha V}\, l_V - c_{\alpha H}\, l_H}{m\, v^2}, \quad a_{22} = -\frac{c_{\alpha V} + c_{\alpha H}}{m\, v}, \quad a_{25} = \frac{c_{\alpha V}}{m\, v}. \tag{21}$$

The parameters are described in Table II. The output of the system is the track error d:

$$\mathbf{y}(t) = (0 \quad 0 \quad 0 \quad 1 \quad 0)^T \mathbf{x}(t). \tag{22}$$

Based on the state space model, the transfer function can easily be determined. The control transfer function is

$$F_c(s) = \frac{i_L}{T_L\, s + 1} \cdot \frac{a_{25} s^2 + (a_{15}\, a_{21} + a_{15} - a_{25}\, a_{11})\, s + (a_{25}\, a_{12} - a_{25}\, a_{12})}{s^2 - (a_{11} + a_{22}) s + (a_{11}\, a_{22} - a_{12}\, a_{21})} \cdot \frac{1}{s} \cdot \frac{v}{s}, \tag{23}$$

and the noise transfer function is

$$F_{\text{noise}} = -\frac{v}{s} \cdot \frac{v}{s}. \tag{24}$$

Table II. Parameters of the bicycle model.

Symbol	Parameter
$c_{\alpha V}$	Skew stiffness, front wheel
$c_{\alpha H}$	Skew stiffness, back wheel
l_V	Wheel base front to center of mass
l_H	Wheel base back to center of mass
θ	Moment of inertia
m	Mass

Parallel structure control As modeled, the vehicle has three degrees of freedom, which are the x and y positions as well as the orientation ψ of the car. Only the steering angle δ is available for controlling the system. As a result, the three degrees of freedom are handled simultaneously. Track error and track angle deviation are used as feedback signals. The working point is chosen at the speed of 30 km/h.

Figure 35 shows the structure of the control strategy used. Again, $K(s)$ stands for each transfer function of the controller. It consists of two parallel control loops for track error and track angle deviation as well as a pilot control taking the curvature of the desired trajectory into consideration. The map-based pilot control algorithm calculates the steering angle that would be needed to follow the desired track based on parameters of the bicycle model.

Performance of the lateral controller Lateral control strategy has to handle different kinds of trajectories. On the one hand, the vehicle has to follow trajectories with a curvature of approximately $\kappa \approx 0$

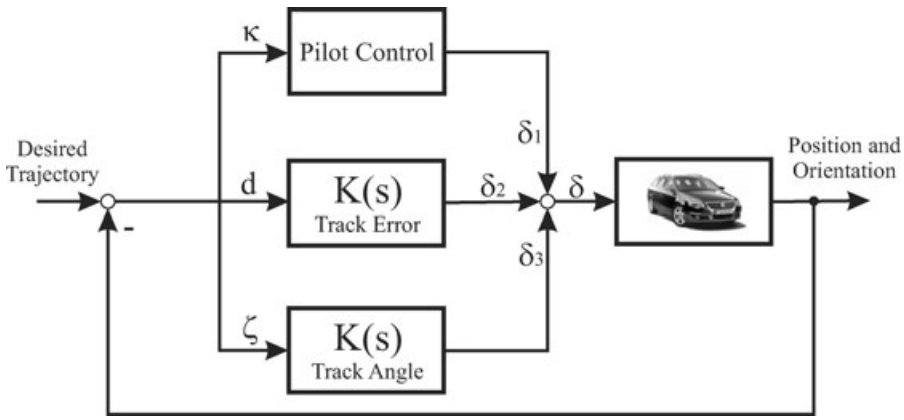


Figure 35. Lateral control strategy.

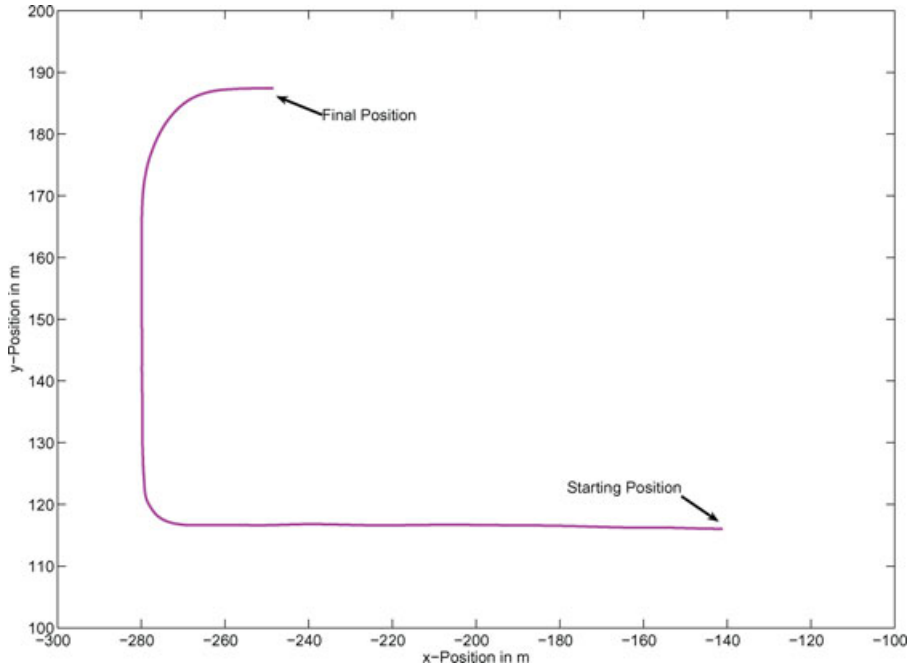


Figure 36. Trajectory.

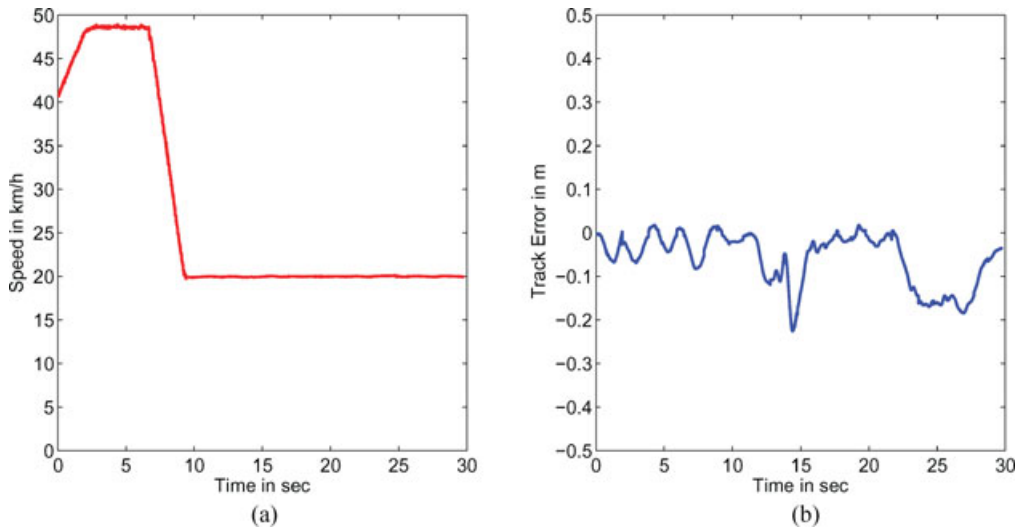


Figure 37. Speed profile (a) and track error (b) of the trajectory.

at higher speeds. On the other hand, the track error in twisting areas is supposed to be as small as possible. Figure 36 shows an example of a trajectory that consists of a long straight part and two sharp curves. On the straight section, the vehicle is accelerated up to a speed of almost $v = 50$ km/h. The curves are driven at a speed of approximately 20 km/h. The speed profile is shown in Figure 37. The performance of the

control strategy in terms of track error can be seen in the same figure.

The control strategy shown worked well during all tests and missions during the DARPA Urban Challenge. It has always been stable with quite a low track error.

4.5. Safety

The safety systems of Caroline have to ensure the highest possible safety for the car and the environment in both manned and unmanned operation. They have to monitor the integrity of all viable hardware and software components. In case of an error, they have to bring the car to a safe stop. Furthermore, they must provide an interface for pausing or disabling the car using a remote E-stop controller. We extended these basic features by including the possibility to reset and restart separate modules independently using hardware and/or software means in order to gain the option of automated failure removal. Figure 38 depicts this watchdog concept.

Caroline is equipped with two separate brake systems: the main hydraulic system and an additional electrical parking brake. The main hydraulic brake is controlled by pressure, usually generated with a foot pedal by the driver. In autonomous mode, this pressure is generated by a small hydraulic brake booster. The parking brake is controlled by a push button in the front console. This brake is a useful additional feature. If the button is pressed while the car is rolling, the main brake system is activated in addition to the parking brake until the car comes to a complete stop. During autonomous mode, the watchdog gateway, the emergency buttons on the top of the car, and the receiver for the remote E-stop controller form a safety circuit, which holds a safety relay open. This relay is connected to the push button for the parking brake. If one of the systems fails or is activated, the safety circuit is opened, the contact of the relay is closed, and the push button of the parking brake is activated. During emergency braking, the lateral controller of the car is still able to hold the car on the given course.

Although the watchdog's main purpose is to ensure safety, it also increases the system's overall reliability. Caroline is a complex system with custom or preproduction hardware and software modules. These components were developed in a very short time and therefore are not as reliable as off-the-shelf commercial products. For this reason we used devices primarily implemented for all safety-relevant subsystems in order to also provide the means to monitor and reset non-safety-relevant subsystems.

Each host runs a local watchdog slave daemon, which monitors all local applications, as shown in Figure 39. A process failing to send periodic heartbeats within a given interval indicates a malfunction, such as memory leakage or deadlocks. Therefore the process and all dependent processes are terminated

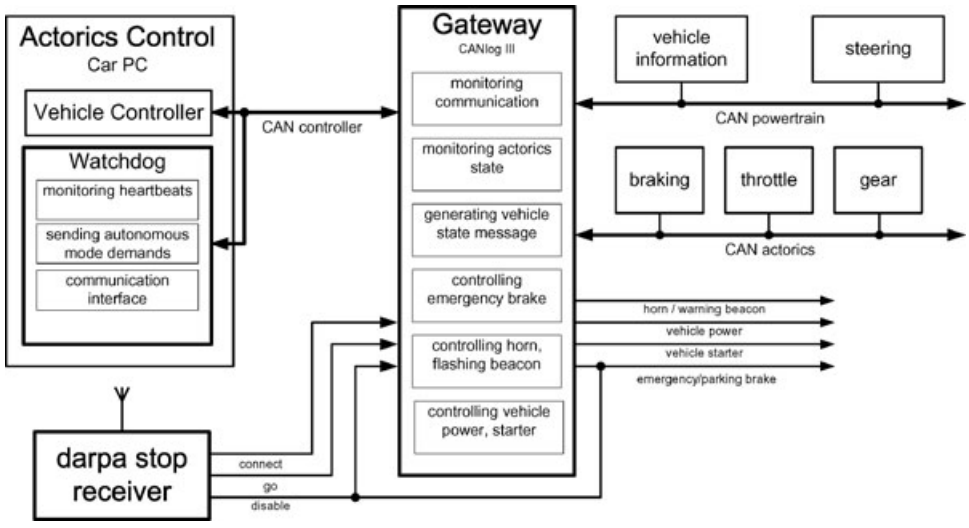


Figure 38. Watchdog architecture.

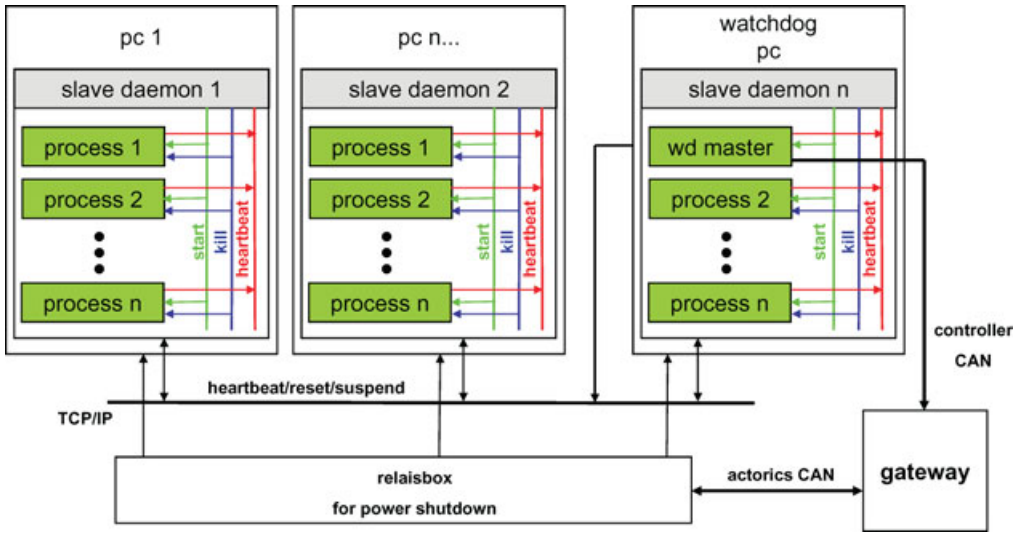


Figure 39. Software watchdog architecture.

by the local watchdog slave, to be restarted with respect to the order required by process dependencies.

The slave watchdog itself is monitored by a remote central master watchdog. This approach allows the detection of malfunctions that cannot be resolved by the local slave watchdog, e.g., if a computer freezes. If a computer should freeze, an emergency stop is initiated and the failed system is power cycled to restart in a stable state. The master watchdog is monitored by the CAN gateway, which initiates an emergency stop on failure of the master watchdog.

5. SYSTEM DEVELOPMENT PROCESS

For developing Caroline's software and ensuring its quality, we implemented a multilevel testing process using elements of extreme programming (Beck, 2005) partly realized in an integrated tool chain shown in Figure 40. The workflow for checking and releasing software formally consists of five consecutive steps. First the source is compiled to check for syntactical errors. While running the test code, the memory leak checker valgrind (Nethercote & Seward, 2003) checks for existing and potential memory leaks in the source

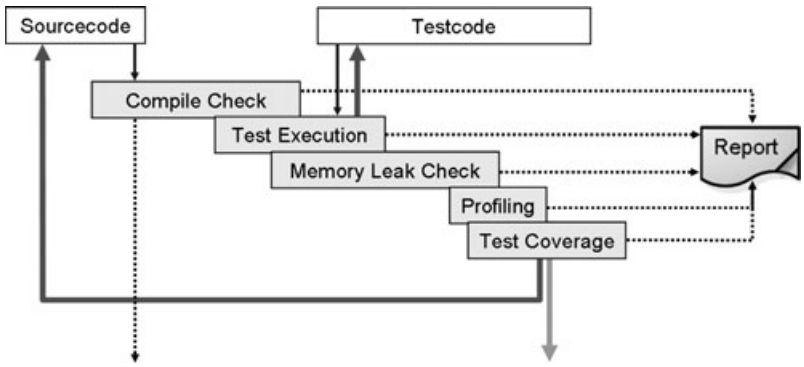


Figure 40. Workflow for testing and releasing software.

code. After the execution of the test code, source code coverage is computed by simply counting the executed statements. The intent is to implement test cases that completely cover the existing source code or to find important parts of the source code that are still lacking test cases. The last step is for optimization purposes only and executes the code in order to find time-consuming parts inside an algorithm.

The tool chain is executed manually by the developer or by using an integrated development environment such as Eclipse. The tool chain itself can be customized by the developer by selecting only necessary stages for the current run, i.e., skipping test suites for earlier development versions of an algorithm. Nevertheless, the complete tool chain is executed every time a new version of the source code is checked in the revision system Subversion (Collins-Sussmann, Fitzpatrick, & Pilato, 2004). Therefore, an independent bugbuster server periodically checks for new revisions on the server. If a new version is found, it is checked out into a clean and safe environment so that the complete tool chain can be run. The results are collected, and a report is automatically generated. The report is easily accessible through the project's Web portal (Edgewall Software, 2007) for every developer. For measuring the performance or consulting the results of a previous revision, the history of older revisions is kept and accessible via the same Web portal.

The main development process described above mainly covers only unit tests (Liggesmeyer, 2002) for some functions or parts of the complete software system. For the development of Caroline's AI, interactive feedback tests using riskless simulations are necessary. Furthermore, the interactive simulations describe different situations for testing the AI. After completing the interactive tests, they can be formalized in acceptance tests for automatic execution on another independent server. These test suites are automatically executed after every change to the revision system, comparable to the bugbuster server.

The next section describes the simulator development for the CarOLO project. Afterward, the adoption of the simulator in automatic acceptance tests is explained. This work continues prior work presented in Basarke, Berger, Homeier, and Rumpe (2007a) and Basarke, Berger, and Rumpe (2007b).

5.1. Simulator

The simulation of various and partly complex traffic situations is the key for developing a high-quality AI that is able to handle many different situations with different types of preconditions. The simulator provides appropriate feedback to the other parts of the system by interpreting the steering commands and changing the ego state and the surroundings.

The simulator can be used for interactively testing newly developed AI functions without the need for a real vehicle. A developer can simply, safely, and quickly test the functions. Therefore, our approach is to provide a simulator that can reliably simulate missing parts of the whole software system. Furthermore, the simulator is also part of an automatic test infrastructure described in the next section.

Figure 41 shows the main classes of the core simulator. The main idea behind this concept is the use of sets of coordinates in a real-world model as context and input. These coordinates are stored in the model and used by the simulator. Every coordinate in the model is represented by a simulator object position describing the absolute position and orientation

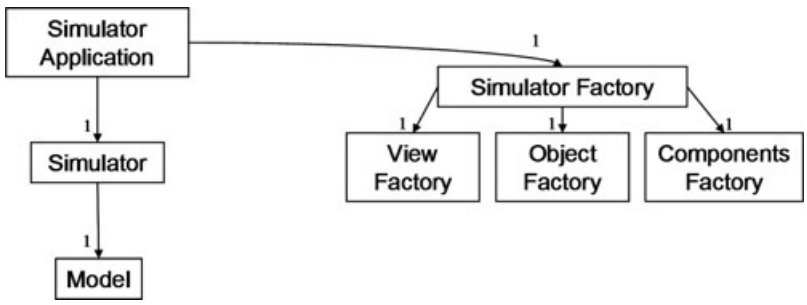


Figure 41. Main classes of the simulator.

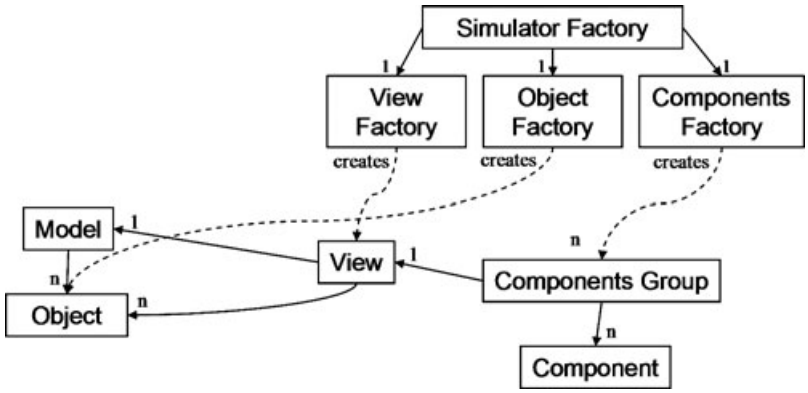


Figure 42. Object factories creating the simulator's surroundings.

in the world. Every position is linked to a simulator object that represents one single object. These objects can have a variety of behaviors, shapes, and other information necessary for the simulation. The model is linked with a simulator control that supervises the complete simulation. The simulator application itself controls the instantiation of every simulator component by using object factories.

Figure 42 shows the factories in detail. The simulator view encapsulates a read-only view of an extract of the world model. Every simulator view is linked with a simulator components group. A component represents missing parts of the whole system, such as an actorics module for steering and braking or a sensor data fusion module for combining measured values and distributing the fused results. Thus, every component in the components group can access the currently visible data of the core data model by accessing the simulator view. As mentioned above, every simulator object position is linked with a simulator object, each of them equipped with its own configuration. Thus, every component can retrieve the relevant data of the owned simulator object.

The main task of the simulator is to modify the world model over time. For simulating the world it is necessary to proceed a step in the simulation. A simulation step is a function call to the world model, with the elapsed time step $\delta t_i > 0$ as a parameter that modifies the world model either sequentially or in parallel.

A simple variant is to modify every simulator object sequentially. In this variant, the list of simulator objects is addressed through an iterator and then modified using original object data. Although this is an efficient approach, it is not appropriate when the objects are connected and rely on behaviors from other objects. Another possibility is to use the algorithm as if a copy of the set of simulator object positions was created. While reading the original data, the modification uses the copy and thus allows a transaction such as a stepwise update of the system, in which related objects update their behavior together.

For modifying an object in the world model, every nonstatic object in the world model uses an object that implements the interface MotionBehavior as shown in Figure 43. A motion behavior routine executes a simulation step for an individual object. A simulator component implementing a concrete motion behavior registers itself with the simulator object. For every simulation step the simulator object must call the motion behavior and therefore enables the behavior implementation to modify its own position and orientation according to a simulator

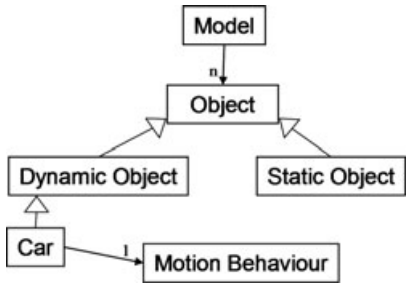


Figure 43. World's model and motion behavior interface.

component. The decoupling of objects and their motion behavior allows us to change the motion behavior during a running simulation, i.e., because of weather influences. Furthermore, it simplifies the implementation of new motion behaviors at development time. For testing Caroline, we developed additional motion behaviors such as MotionBehaviorByKeyboard for controlling a virtual car in the interactive mode by using keys or a MotionBehaviorByRNDF that controls a car in its surroundings by using a predefined route to follow.

The most interesting motion behavior, however, is the MotionBehaviorByTrajectory because it communicates directly with the AI. For the best imitation of the behavior of the real car, the simulator uses the same code as the vehicle control module based on trajectories expressed as a string of pearls that form consecutive gates. Furthermore, the motion of the simulated car is computed with third-order B-splines such as the vehicle controller module. Using a B-spline yields smoother motion in the simulation and a driving behavior sufficiently close to reality, if it is taken into account that for intelligent driving functions it is not necessary to handle the physical behavior in every detail, but in an abstraction useful for an overall correct behavior.

Using motion behaviors, it is possible to compose different motion behaviors to create a new composed motion behavior. For example, it is possible to build a truck with trailer from two related but only loosely coupled objects. A composition of the motion behaviors yields a new motion behavior that modifies the position and orientation of the related simulator objects according to inner rules as well as general physical rules.

Getting such a simulator up and running requires quite a number of architectural constraints for the software design. One important issue is that no component of the system being tested tries to call any system functions directly, such as threading or communication, but only through an adapter. Depending on whether it is a test or an actual running mode, the adapter decides whether the function call is forwarded to the real system or substituted by a result generated by the simulator. Because of the architectural style, it is absolutely necessary that no component retrieve the current time by calling a system function directly. Time is fully controlled by the simulator, and therefore it knows which time is relevant for a specific software component if different times are used. Otherwise, time-based algorithms will become confused if different time sources are mixed up.

5.2. Quality Assurance

As mentioned at the beginning of this section, the simulator is used not only for interactive development of the AI. It is also part of a tool chain that is automatically executed on an independent server for ensuring the quality of the complete software system consisting of several modules. In the CarOLO project, we analyzed the DARPA Urban Challenge documents to understand the requirements. These documents contained partly functional and nonfunctional definitions for the necessary vehicle capabilities. In every iteration a set of tasks consisting of new requirements and bugs from previous iterations is chosen by the development team, prioritized and concretely defined using the Scrum process for agile software engineering (Beedle & Schwaber, 2002). These requirements are the basis for both a virtual test drive and a real test of Caroline.

After designing a virtual test drive, the availability of necessary validators is checked. A validator is part of the acceptance tool chain and responsible for checking the compliance of the AI's output with the formal restrictions and requirements. Validators implementing intelligent software functions are used to automatically determine differences in the expected values in the form of a constraint that cannot be violated by the test. A validator implements a specific interface that is called up automatically after a simulator step and right before the control flow returns to the rest of the system. A validator checks, for example, distances to other simulator objects or validates whether a car has left its lane or exceeded predefined speed limits. After an unattended virtual test drive, a boolean method is called upon to summarize the results of all test cases. The results are collected and formatted in an e-mail and Web page for the project's Web portal.

The set of validators covers all basic requirements and restrictions and can be used for automatically checking the functionality of new software revisions. The main benefit is that these high-level tests are black-box tests and do not rely on the internal structure of the code. Thus, a subgroup of the CarOLO team was able to develop these high-level acceptance tests without a deep understanding of the internal structures of the AI. Using this approach,

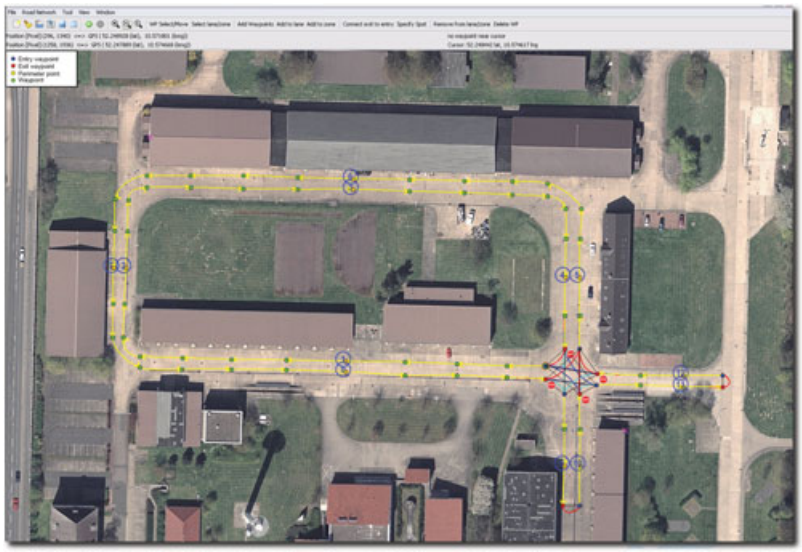


Figure 44. Screenshot of the GUI tool for constructing RNDFs.

more-complex traffic situations could be modeled and repeatedly tested without great effort.

To allow for the quick and convenient creation of test scenarios, various concepts and tools have been developed. The following describes how virtual test drives are defined as well as how certain surroundings such as data fusion objects or drivability data are generated and fed into the simulator. To make this clear, we briefly illustrate the proceedings on the basis of an example, which deals with the simple passing maneuver as described in Section 4.3.3. Assume that we would like to determine whether the AI is able to recognize static obstacles in our travel lane and react properly by adhering to the required minimum distances.

First, an RNDF must be created that contains information about existing lanes, intersections, parking spots, and their connections. As an RNDF provides the basis for every test run, many of those route network definitions had to be created. Therefore we developed a graphical user interface (GUI) tool to simplify the creation of RNDFs as shown in Figure 44.

Several features including dragging way points, connecting lanes, and adding stop signs or checkpoints speed up the construction process. Completed RNDFs could be exported to a text file and used as input for the AI as well as for the simulator.

The purposes of RNDFs within the simulator vary in different ways. One purpose is to check the behavior of the AI concerning the RNDF provided and the actual lane. Therefore a second RNDF can be passed to the simulator. The additional and independent RNDF is used to provide lane data, which are normally detected by the computer vision system. This is especially important if there are major differences between the linear distance and the actual route to the next way point.

Another use of RNDFs is to define the behavior of dynamic obstacles during the test run, as mentioned earlier. Thus we are able to check relevant software modules for their interaction with dynamic obstacles. This approach is similar to the one used for providing detected lanes. Dynamic obstacles are interacting on a basis of their individual RNDFs by using the MotionBehaviorByRNDF. This concept can be used for simulating scenarios at intersections and even more complex traffic scenarios.

To extend the example of passing a static obstacle, we need to create suitable data, which could be translated to sensor fusion objects. Two principal approaches are available to achieve this goal. Generating scenarios with static obstacles can be accomplished by using our visualization application, which provides the ability to define polygons or by using a drawing tool. Shapes of fusion objects could be exported to a comma-separated file. The simulator parses the textual representation of polygons and translates them to fusion objects to be processed by

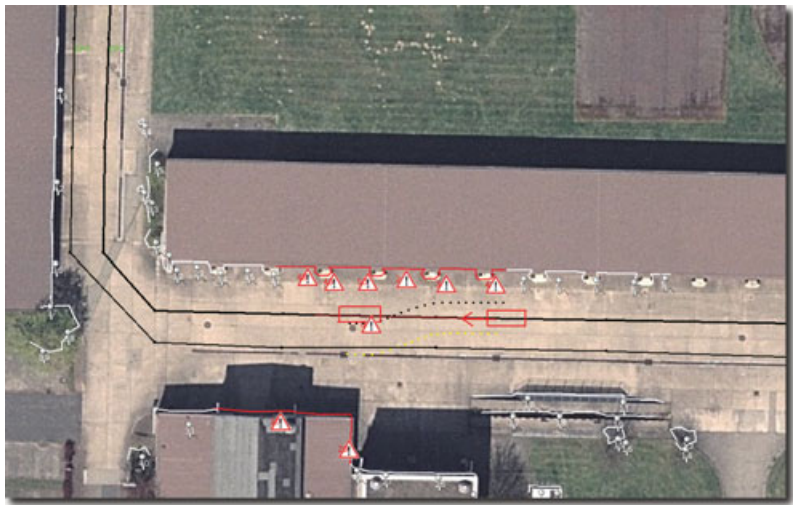


Figure 45. Screenshot with fusion objects.

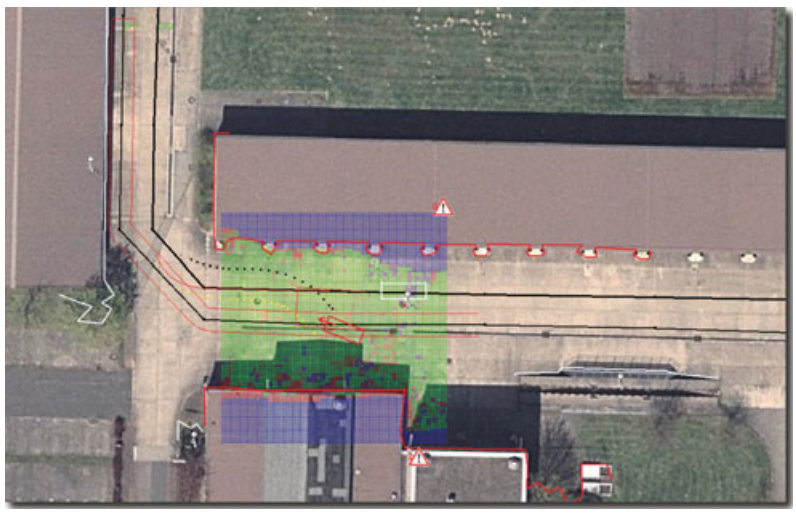


Figure 46. Screenshot with additional drivability data.

the AI. The use of a drawing tool implies the use of predefined colors. The positions of static obstacles are computed by scanning the created image for special markers with reference to a known coordinate. Figure 45 depicts a screenshot of our visualization application where the corresponding fusion objects are displayed.

For a more realistic simulation, the data fusion objects generated by the simulator could be created with a different quality. This is used to simulate sensor noise and GPS drifts and makes fusion objects suddenly disappear or moves them by a tiny offset away from their original location. The sensor visibility range could be specified to affect the range of fusion objects that will be transmitted to the AI.

Adding moderate drivability data completes this test run. This could be accomplished by passing an image file to the simulator, which specifies the required information through different colors. Figure 46 shows the result. The visualization of a

drivability grid displays drivable terrain in green, undrivable terrain in red, and unknown terrain with blue cells.

6. THE RACE AND DISCUSSION

6.1. National Qualification Event

The NQE took place from October 26 to 31 on the former George Air Force Base in Victorville, California, as depicted in Figure 2. The entire area was divided into three major parts named Area A, Area B, and Area C, as shown in Figure 47. First, Caroline had to demonstrate the proper function of her safety system to participate in the NQE. As expected, Caroline stopped within the necessary range using the E-stop remote controller as well as the emergency stop buttons mounted on her roof.

6.1.1. Area A

For our team, the NQE started in Area A. The main task for Caroline in that part was to merge into and through moving traffic. Therefore, several other vehicles controlled by human drivers drove within predefined speed limits to ensure the 10-s time slots as demanded by the DARPA's requirements. Figure 48 shows the layout of the track. Caroline was placed at checkpoint 2. She had to drive downward to the T-junction, wait for an appropriate time slot, and then turn left through the moving traffic. Afterward, she had to pass checkpoint 1 by following other vehicles and drive to the upper junction. After waiting for an appropriate time slot, she had to turn into the street to pass checkpoint 2 again. The goal was to drive as many rounds as possible in this area.

Compared to other competitors, Caroline had to pass this task several times. The first run in this part let Caroline drive into the opposite lane. Analyzing this obviously strange behavior afterward using our simulator as depicted in Figure 49, we figured out that the barriers shown by white lines around the course narrowed the proper lane. Therefore, Caroline, shown as a red rectangle driving downward to the lower T-junction, interpreted them as stationary obstacles in her way, which she tried to overtake, which can be seen in the computed trajectory shown by yellow and black pearls that leads into the opposite lane.

After modifying several parameters, we had our second try in Area A. She drove five rounds, merged into moving traffic correctly, waited at stop lines, and followed other vehicles very well. Unfortunately, some problems occurred on the above right corner,

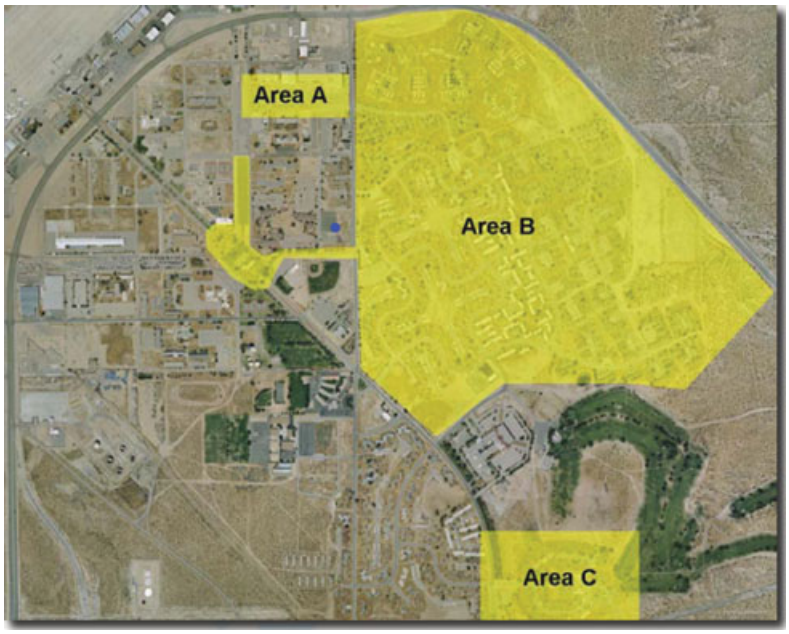


Figure 47. Layout of the former George Air Force Base for the NQE. The blue dot indicates the pit area for our team.

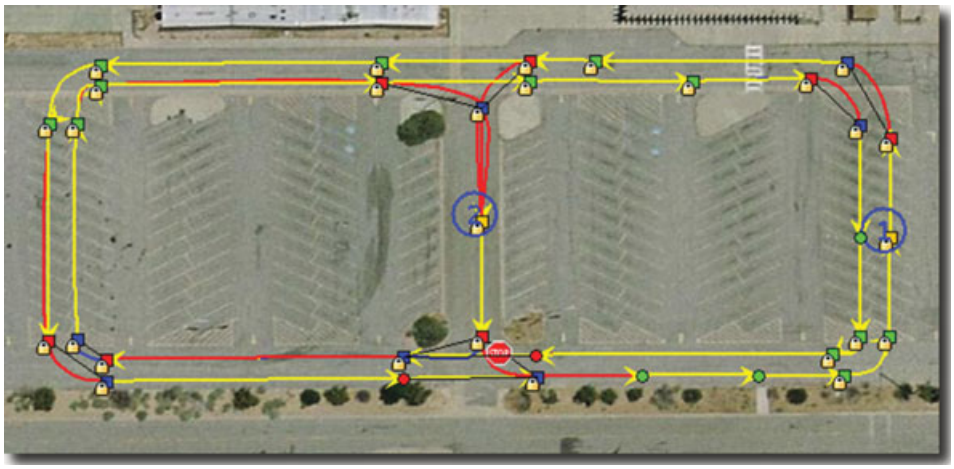


Figure 48. Layout of Area A.

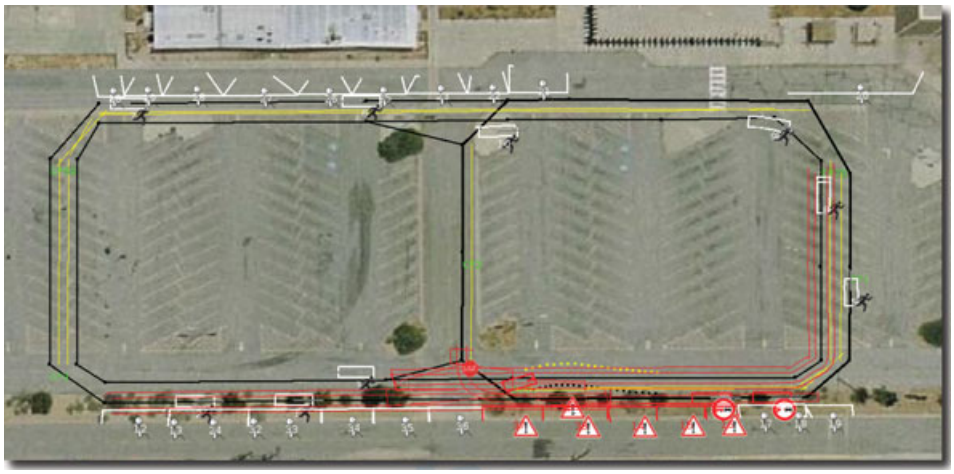


Figure 49. Analysis of Caroline's behavior in Area A.

when Caroline decided to turn right instead of following the road to the junction. We found out that Caroline got in trouble with the street surface in that corner. There was a mixture of concrete and tar, each with different colors. Thus, Caroline studied that color difference and tried to drive toward areas with a similar surface.

After modifying that behavior, we got another try in that course. Caroline started a perfect first run but waited too long for the second one. Therefore, the judges paused our vehicle and demanded a more progressive behavior of Caroline. Tuning again some parameters, we tried the course a fourth time a short time later. This time, Caroline drove very swiftly but she did not give way to oncoming traffic. So, we changed the parameters again to get a safer behavior and convinced the judges in our last try in that area of Caroline's abilities to merge correctly into moving traffic after demonstrating approximately eight perfect rounds.

6.1.2. Area B

After encountering difficulties in the first task, we were unsure how Caroline would perform in Area B because several teams already failed to complete

Figure 50. Layout of Area B.

this part. The entire course is shown in Figure 50. The main task was to overtake stationary obstacles, handle free navigation zones without any lane markings, and park safely inside those zones between other vehicles. The course itself could not be seen completely, so Caroline had to drive for herself without any observation by our team. We could hear her progress only by the team radio and by her siren.

Caroline started within a concrete start chute laid inside a free navigation zone. Many other teams already failed to leave this zone into the traffic circle correctly. She smoothly entered the traffic circle, left the circle, and turned into the part on the right-hand side of Figure 50. In the center of the lower circle, she had to park between other vehicles. The entry to that zone was very rough, and several other teams had already damaged the tires of their vehicles. We analyzed the video right after the task and remarked heavy vibration of the camera's picture, but she entered the zone smoothly. After finishing the parking she left the zone to proceed on the course.

Furthermore, Caroline had to deal with a gate located right at the exit of the upper circle. Owing to our sensor layout, she had to attempt several times to find the right way for leaving that circle. Returning to the start chutes again, she honked twice to indicate the completion of her mission after passing the last checkpoint. With this successful run, Caroline was one of only three vehicles to accomplish this course completely and on time.

6.1.3. Area C

On the same day, Caroline was faced with Area C. This area is shown in Figure 51. The main task was to handle intersections correctly and deal with blocked roads.

Caroline started near checkpoint 30 in the upper left-hand corner on the outer lane. She handled both intersections on the left-hand side and the right-hand side several times correctly with every combination of other vehicles she faced. Right in front of checkpoint 30 in the center part of this course, Caroline encountered a road blockage, as shown in Figure 52. We were unsure whether Caroline would detect the barrier because it had no contact to the ground and our sensors could look right through that barrier. But Caroline detected that

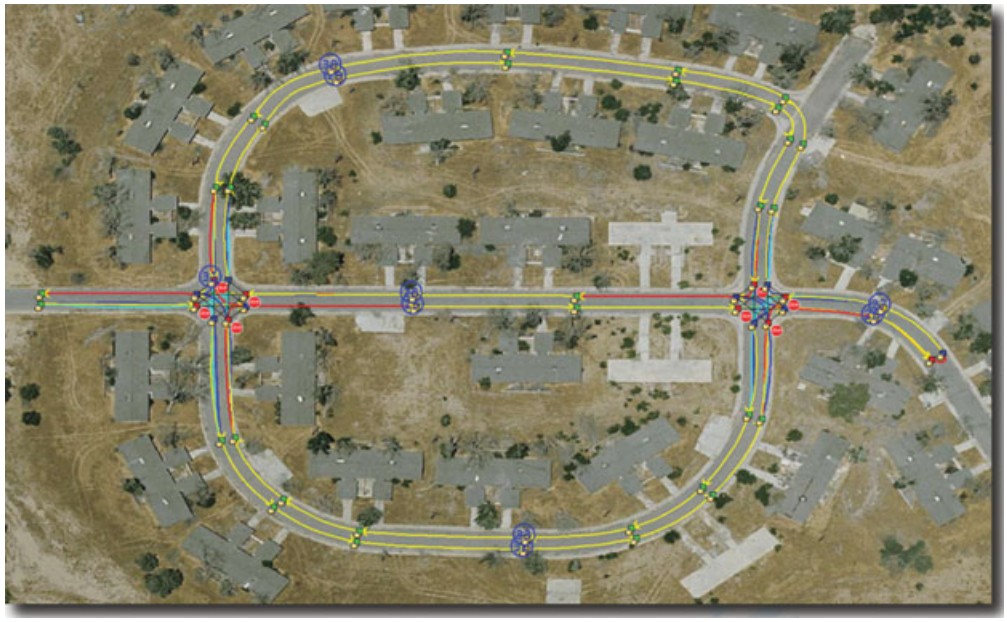


Figure 51. Layout of Area C.

Figure 52. Blocked round in Area C by a barrier.

barrier properly and initiated the U-turn to choose another route to the checkpoint. Afterward, she passed all further traffic and intersection situations correctly and finished Area C finally. With all results achieved in the three areas, Caroline qualified early as a newcomer for the final event along with the well-established teams with their Grand Challenge experience.

6.2. Mandatory Practice for DARPA Urban Challenge Final Event

The day before the DARPA Urban Challenge Final Event was scheduled, each of the 11 finalists had to participate in a practice session. By using this session, DARPA would ensure that every vehicle was able to leave the start chute and turn into the traffic circle. Assuming that this would be an easy task, we put Caroline into autonomous mode and waited for her to begin her run. But she did not leave her start chute, and our team failed that practice session. We figured out the problem by parsing the RNDF provided by DARPA. This issue did not let Caroline understand the road network for the final. After fixing this problem, we got another try. But Caroline still did not leave her start chute. Thus, DARPA placed us in the last of the 11 start chutes and canceled the practice for our team.

Later analyzing the data, we figured out that there was a jitter in the GPS signal while significantly waiting for the RUN mode that yielded leaving the calculated trajectory. After fixing this, we finally prepared Caroline for the DARPA Urban Callenge Final Event on the following day.

6.3. DARPA Urban Challenge Final Event

Figure 53 shows the enlarged Area B track for the DARPA Urban Challenge Final Event, including the former Area A as a parking lot. The start chutes were the same as for the run in Area B. Additionally, in the lower right corner of the map, there was a sandy offroad track located yielding a two-lane road returning to the inner part of the DARPA Urban Challenge Final Event area.

On November 3, 2007, at 6:53 a.m. PST we loaded the first of three mission files into Caroline and set her into PAUSE mode. She calculated the route for the first checkpoint and started her run at 7:27 a.m. PST. Figure 54 shows the first part of her way during the first mission.

The asterisk in Figure 54 indicates the location where two members of our team had to accompany the DARPA judges. Caroline had passed approximately 2.5 km before she was paused by the DARPA control vehicle right behind her. Figure 55 shows the reason for the PAUSE mode.

Caroline got stuck after she turned into the berms. Figures 55(a) and 55(b) show Caroline approaching a traffic jam right in front of her. Obviously, she tried to pass the stopped vehicle by interpreting it as a stationary obstacle using the clearance to the last car. The result of this attempt is shown in Figure 55(c): Caroline got stuck and could not get free without human intervention.

After she got freed and set in RUN mode again right at the beginning of the two-lane road, she continued her route and passed several checkpoints

Figure 53. Layout for the DARPA Urban Challenge Final Event.

Figure 54. Passed way before the first problem.

(a) Blocked road in front of Caroline

(b) Caroline approaching the traffic jam

(c) Caroline stuck after turning left into the berms

Figure 55. Caroline got stuck after 2.5 km.

(Figure 56). The next incident was after 11.4 km, shown as the asterisk in Figure 57.

At that location Caroline did not yield right-of-way to Talos, the autonomous vehicle from team MIT. Therefore, DARPA paused both vehicles and let team members from MIT come to that location. After replacing Talos, both vehicles were sequentially set to RUN mode and safely passed each other. Unfortunately, the reason for not yielding right-of-way to Talos could not be figured out by analyzing our log files. Because the situation was a left turn through oncoming traffic, it could be a problem detecting and tracking Talos due to problems either with our front sensors or with the interpretation in the AI.

As shown in Figure 58, Caroline continued her route. Additionally, she parked in the parking lot shown in the upper left picture of Figure 58. After the parking maneuver, she returned a second time to the traffic circle and continued her mission 1.

At approximately 9:55 a.m. PST, two team members from team CarOLO were again driven to Caroline, who met Talos from team MIT for the second time in a free navigation zone. This incident is shown as an asterisk in Figure 59.

Our team members were faced with a twisted carrier rod of the Ibeo laser scanners due to a collision with Talos from team MIT, as shown in Figure 60. It is still unresolved which car was the cause of the accident. Caroline interpreted the situation as described in the technical evaluation criteria (DARPA, 2006) in the section "Obstacle field." Therefore, Caroline tried to pass the oncoming Talos by

Figure 56. Caroline went on after she got stuck.

Figure 57. Next incident including Caroline and Talos from team MIT.

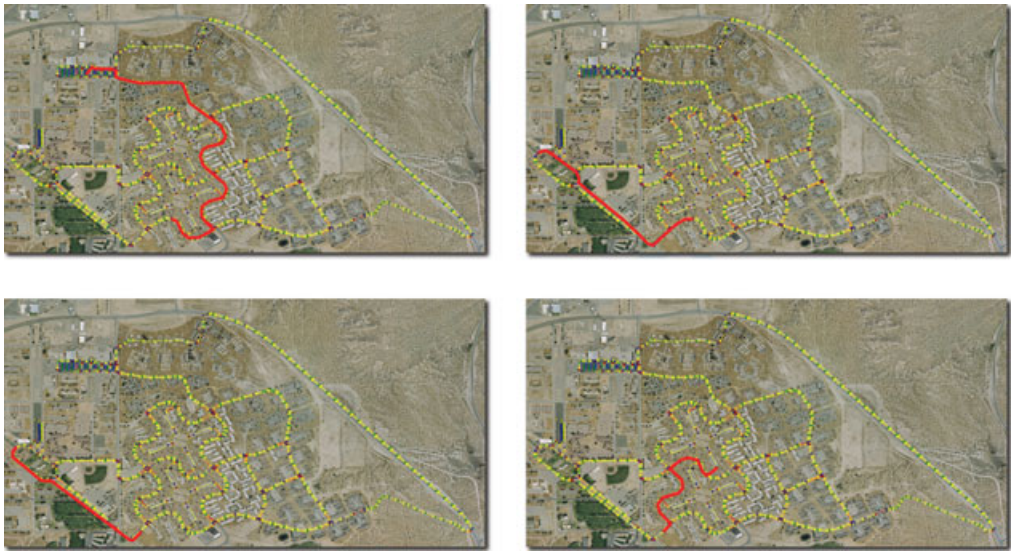


Figure 58. Caroline went on after not yielding right-of-way to Talos.

Figure 59. Passed way before the second problem.

pulling to the right. Unfortunately, further interpretation is impossible due to missing detailed log files of that situation. Finally, DARPA retired Caroline as the fourth and last vehicle from the DARPA Urban Challenge Final Event.

Altogether, Caroline drove 16.4 km and was retired from the race at 10:05 a.m. PST. At 8:03 a.m. PST, the watchdog module reset the SICK laser scanners mounted on the roof due to communication problems. At approximately 9:00 a.m. PST, the watchdog missed heartbeats from the IMU and therefore triggered a reset. Right after the collision with Talos from team MIT, the watchdog observed communication problems with the laser scanners mounted in the front of Caroline. After a reset, the communication was reestablished. During the race, computer "Daq1"

(a) Collision between Caroline and Talos from MIT

(b) Red circle shows the twisted carrier rod

Figure 60. Caroline was retired after the collision with MIT.

as shown in Figure 4 froze two times and had to be reset.

7. CONCLUSION

Team CarOLO is an interdisciplinary team made up of members from faculties of computer science and mechanical and electric engineering that are significantly supported by industrial sponsors. Our vehicle Caroline is a standard 2006 Volkswagen Passat station wagon built to European specifications that is able to detect and track stationary and dynamic obstacles at a distance of up to 200 m. The system's architecture comprises eight main modules: sensor data acquisition, sensor data fusion, image processing, digital map, AI, vehicle path planning and low-level control, supervisory watchdog and online-diagnosis, and telemetry and data storage for offline analysis. The signal flow through these modules is generally linear in order to decouple the development process. Our design approach uses a multisensor fusion of LIDAR, radar, and laser scanners, extending the classical point shape–based approach to handle extensive dynamic targets expected in urban environments. Image processing detects lane markings along with drivable areas. AI is modeled according to DAMN architecture, redesigned and enhanced to meet requirements of special behavior in urban environments. Our approach is able to handle complex situations and ensure Caroline's proper behavior, e.g., obeying traffic regulations at intersections or performing U-turns when roads are blocked. Decisions of the AI are sent to the path planner, which calculates optimal vehicle trajectories with respect to its dynamics in real time. Safety and robustness are ensured by supervisory watchdog monitoring of all the vehicle's hardware and software modules. Failures or malfunctions immediately result in a safe and complete stop by Caroline. Because we are a large heterogeneous team with a very tight project schedule, we recognized very early the need for efficient quality assurance during the development process. Thus, we implemented an automatic multilevel test process. Each new feature or modification runs through a series of unit tests or comprehensive simulations before being deployed on the vehicle.

As a competitor in the DARPA Urban Challenge Final Event, Caroline was able to autonomously perform missions in urban environments. She drove approximately 17 km in about 3 h in the final.

ACKNOWLEDGMENTS

The authors thank their colleagues, students, and professors from five institutes of the Technische Universität Braunschweig, who developed Caroline. As a large amount of effort and resources were necessary to attempt this project, it would not have been successful if not for the many people from the university and local industry that sponsored material, manpower, and financial assistance. Particular thanks go to Volkswagen AG, IAV GmbH, and the Ministry of

Science and Culture of Lower Saxony. The authors' team also thanks Dr. A. Bartels, Dr. H. Hoffmann, Professor J. Hesselbach, Mr. F. Horch, Mr. N. Lange, Professor J. Leohold, Dr. H. Lienkamp, Mr. T. Kuser, Professor U. Seiffert, Mr. C. Spichalsky, Professor J. Varchmin, Professor E. Wand, and Mr. U. Wehner for their help on various occasions.

REFERENCES

Basarke, C., Berger, C., Homeier, K., & Rumpe, B. (2007a). Design and quality assurance of intelligent vehicle functions in the "virtual vehicle." In Virtual Vehicle Creation 2007 (pp. 131–142). Wiesbaden, Germany: Vieweg.

Basarke, C., Berger, C., & Rumpe, B. (2007b). Software & systems engineering process and tools for the development of autonomous driving intelligence. Journal of Aerospace Computing, Information, and Communication, 4, 1158–1174.

Beck, K. (2005). Extreme programming explained: Embrace change. Boston: Addison Wesley.

Beedle, M., & Schwaber, K. (2002). Agile software development with scrum. Upper Saddle River, NJ: Prentice Hall.

Bilmes, J. (1997). A gentle tutorial on the EM algorithm and its application to parameter estimation for gaussian mixture and hidden Markov models (Tech. Rep. TR-97-021). Berkeley, CA: International Computer Science Institute.

Bradski, G., Kaehler, A., & Pisarevsky, V. (2005). Learning-based computer vision with Intel's open source computer vision library. Intel Technology Journal, 126–139.

Collins-Sussmann, B., Fitzpatrick, B. W., & Pilato, C. M. (2004). Version control with subversion. O'Reilly Media.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2002). Introduction to algorithms, 2nd ed. Cambridge, MA: MIT Press.

DARPA (2006). Technical evaluation criteria. http://www.darpa.mil/grandchallenge/docs/Technical_Evaluation_Criteria_031607.pdf. Accessed July 29, 2008.

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

Edgewall Software (2007). Trac. Edgewall Software. http://www.edgewall.com. Accessed July 29, 2008.

Heikkilä, J., & Silvén, O. (1996). Calibration procedure for short focal length off-the-shelf CCD cameras. In 13th International Conference on Pattern Recognition, Vienna, Austria (pp. 166–170). Heidelberg, Germany: Akademischer Verlag.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. Journal of Basic Engineering, 82, 35–45.

Liggesmeyer, P. (2002). Software-Qualitaet: Testen, Analysieren und Verifizieren von Software. Spektrum, Akad. Verl.

Nethercote, N., & Seward, J. (2003). Valgrind: A program supervising framework. Theoretical Computer Science, 89, 23.

OpenCV Website (2007). The Open CV library. http://www.opencv.org. Accessed July 29, 2008.

Pitteway, M. L. V. (1967). Algorithm for drawing ellipses or hyperbolae with a digital plotter. Computer Journal, 10(3), 282–289.

Rosenblatt, J. (1997). DAMN: A distributed architecture for mobile navigation. Ph.D. thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Shafer, G. (1976). A mathematical theory of evidence. Princeton, NJ: Princeton University Press.

Shafer, G. (1990). Perspectives on the theory and practice of belief functions. International Journal of Approximate Reasoning, 3, 1–40.

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.-E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., & Mahoney, P. (2006). Stanley: The robot that won the DARPA Grand Challenge. Journal of Field Robotics, 23, 661–692.

Ulrich, I., & Nourbakhsh, I. (2000). Appearance-based obstacle detection with monocular color vision. In Proceedings of the AAAI National Conference on Artificial Intelligence, Austin, TX (pp. 866–871).