7

The Golem Group / UCLA Autonomous Ground Vehicle in the DARPA Grand Challenge

Richard Mason, Jim Radford, Deepak Kumar, Robert Walters, Brian Fulkerson, Eagle Jones, David Caldwell, Jason Meltzer, Yaniv Alon, Amnon Shashua, Hiroshi Hattori, Emilio Frazzoli, and Stefano Soatto

The Golem Group, 911 Lincoln Boulevard #7, Santa Monica, California 90403

Summary. This paper presents the Golem Group/UCLA entry to the 2005 DARPA Grand Challenge competition. We describe the main design principles behind the development of Golem 2, the race vehicle. The subsystems devoted to obstacle detection, avoidance, and state estimation are discussed in more detail. An overview of the vehicle performance in the field is provided, including successes together with an analysis of the reasons leading to failures.

7.1 Overview

The Golem Group is an independent team of engineers formed to build a vehicle for the 2004 DARPA Grand Challenge (DGC). For the 2005 DARPA Grand Challenge, the Golem Group and UCLA's Henry Samueli School of Engineering and Applied Science joined forces to build a second autonomous vehicle, Golem 2. Performance highlights of this vehicle are summarized in Table 7.1.

The novel aspect of the DGC was to require high-speed autonomous driving in the unstructured or semi-structured environment typical of rough desert trails. GPS waypoint-following was necessary, but not sufficient, to traverse the route, which might be partially obstructed by various obstacles. In order to have a good chance of completing the course, vehicles needed to drive much faster, yet have a lower failure rate, than previously achieved in an off-road environment without predictable cues. High-speed driving over rough ground posed a vibration problem for sensors.

7.1.1 Relation to Previous Work

Prior to the DGC, unmanned ground vehicles had driven at very high speeds in structured, paved environments [Dickmanns, 1997, 2004]. Other vehicles had operated autonomously in unstructured off-road environments, but generally not at very high speed. Autonomous Humvees using ladar to detect obstacles in an off-road environment have been developed at NIST [Coombs et al., 2000, Hong et al., 2000]. The U.S. Army eXperimental Unmanned Vehicle [Bornstein and Shoemaker, 2003] also used ladar to detect obstacles and could navigate unstructured rough ground at somewhat over 6 km/hr. Rasmussen

M. Buehler, K. Iagnemma, and S. Singh (Eds.): DARPA'05, STAR 36, pp. 205–243, 2007.
springerlink.com © Springer-Verlag Berlin Heidelberg 2007

Table 7.1. Golem Group / UCLA performance in the 2005 DARPA Grand Challenge

NQE Performance Highlights	
9 min 32 sec NQE run clearing 49/50 gates and 4/4 obstacles	Only Stanford, CMU, and IVST made faster runs clearing all obstacles Only Stanford and CMU made faster runs clearing at least 49 gates
12 min 19 sec NQE run clearing 50/50 gates and 5/5 obstacles	Only Stanford, CMU, Princeton, and Cornell made faster flawless runs
GCE Performance Highlights	
Peak speed (controlled driving) 47 mph Completed 22 race miles in 59 min 28 sec Anecdotally said to be fastest vehicle reaching 16-mile DARPA checkpoint Crash after 22 miles due to memory management failure	

Fig. 7.1. Golem 2 driving autonomously at 30 miles per hour

[2002] used a combination of ladar and vision to sense obstacles and paths in off-road environments. The Carnegie Mellon University Robotics Institute had perhaps the most successful and best-documented effort in the first DGC, building an autonomous Humvee which was guided by ladar [Urmson, 2005, Urmson et al., 2004].

Fig. 7.2. Golem 1 negotiating a gated crossing in the 2004 DARPA Grand Challenge. (Photos courtesy of DARPA).

Golem 1, our own first DGC entry, used a single laser scanner for obstacle avoidance. Golem 1 traveled 5.1 miles in the 2004 Challenge, before stopping on a steep slope because of an excessively conservative safety limit on the throttle control. This was the fourth-greatest distance traveled in the 2004 DGC, a good performance considering Golem 1's small total budget of $35,000.

Our attempts to improve on the performance of previous researchers were centered on simplified, streamlined design—initially in order to conserve costs, but also to enable faster driving by avoiding computational bottlenecks. For example, we relied primarily on ladar for obstacle avoidance, but unlike the majority of ladar users, we did not attempt to build a 3D model of the world *per se*. Instead we only attempted to detect the most important terrain features and track the locations of those on a 2D map. Golem 1, with its single ladar scanner, may have gone too far in the direction of simplicity, and Golem 2 carried multiple ladars to better distinguish slopes and hillsides. Ladar obstacle detection is further discussed in Section 7.3.

As another example of simplification, our path planning process considers possible trajectories of the truck as simple smooth curves in the 2D plane, with curvature of the trajectory as a measure of drivability and distance to the curve as a proxy for danger of collision. We do not consider obstacles in a configuration

space of three dimensions, much less six dimensions. Our approach might be inadequate for navigating a wholly general, maze-like enviroment, but more importantly for our purposes, it gives fast results in the semi-structured case of a partially-obstructed dirt road. Trajectory planning is discussed further in Section 7.4.

We did experiment with some vision systems in addition to ladar. Mobileye Vision Technologies Ltd. provided Golem 2 with a monocular roadfinding system, which is discussed in Section 7.6.1 and also in Alon et al. [2006]. This could be considered as extending the cue-based paved-environment work of Dickmanns, and of Mobileye, to an unpaved environment. Experiments with a Toshiba stereo vision system are described in Section 7.6.2.

7.2 Vehicle Design

Each of our vehicles was a commercially-available pickup truck, fitted with electrically-actuated steering and throttle, and pneumatically-actuated brakes.

We felt it was very important that the robot remained fully functional as a human-drivable vehicle. Golem 1 seats two people while Golem 2 seats up to five. A passenger operated the computer and was responsible for testing, while the driver was responsible for keeping the vehicle under control and staying aware of the environment. During testing, it was very convenient to fluidly transition back and forth between autonomous and human control. Individual actuators (brake, accelerator, steering) can be enabled and disabled independently, allowing isolation of a specific problem. Having a human "safety driver" increased the range of testing scenarios we were willing to consider. Finally, having a street-legal vehicle greatly simplified logistics.

Accordingly, a central principle behind actuation was to leave the original controls intact to as great an extent as possible, in order to keep the vehicle street legal. The steering servo had a clutch which was engaged by a pushrod that could be reached from the driver's seat. The brakes were actuated by a pneumatic cylinder that pulled on a cable attached, through the firewall, to the back of the brake pedal. The cable was flexible enough to allow the driver to apply the brakes at any time. The pressure in the cylinder could be continuously controlled via a voltage controlled regulator. A servo was attached directly to the steering column.

A second design aim was to keep the computational architecture as simple as possible. The core tasks of autonomous driving do not require a large amount of computational power. We worked to keep the software running on a single laptop computer. Unburdened by a rack full of computers, we were able to retain working space in the vehicle, but more importantly, any team member could plug in their laptop with a USB cable and run the vehicle. A block diagram of the architecture is shown in Figure 7.3.

7.2.1 Sensors

The sensors mounted on Golem 2 for vehicle state estimation included a Novatel ProPak LBPlus differential GPS receiver with nominal 14-cm accuracy using

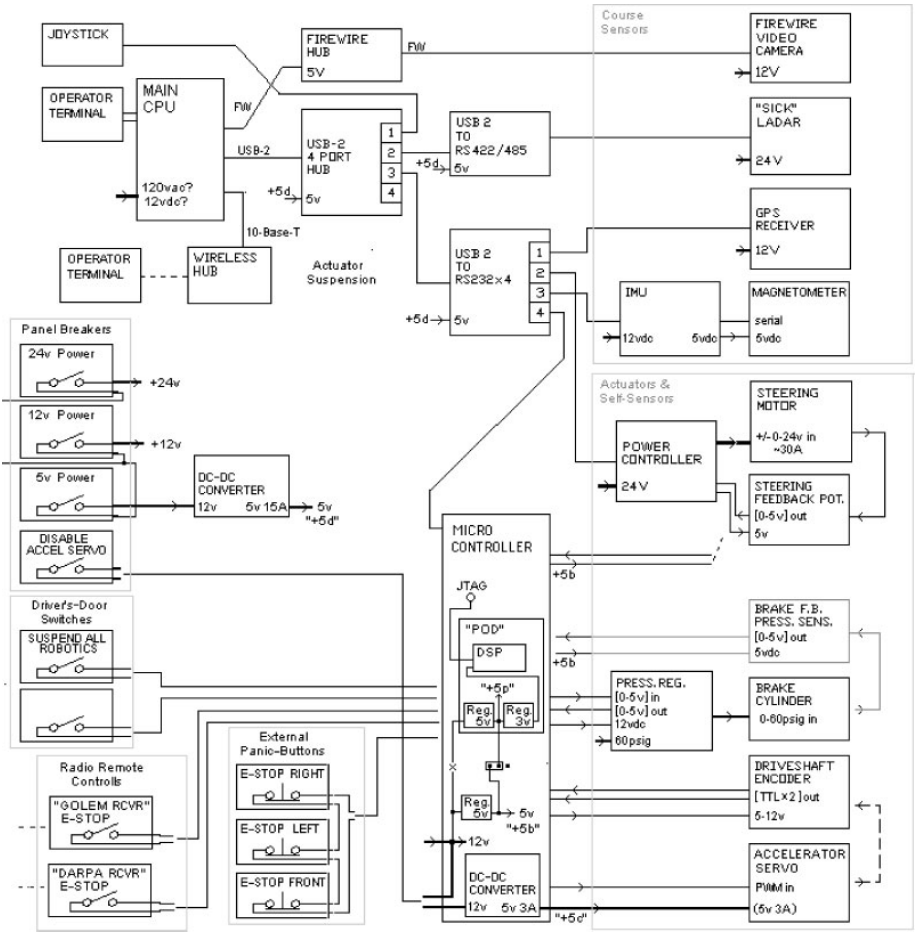

Fig. 7.3. Block diagram of the Golem vehicle architecture

OmniStar HP correction; a BEI C-MIGITS inertial measurement unit (IMU); a custom Hall encoder on the differential for odometry with approximately 10-cm accuracy; and a 12-bit encoder for measuring the steering angle. Vehicle state estimation is discussed further in Section 7.5.

The sensors used for terrain perception included a Sick LMS-221 ladar, which swept a 180-degree arc in front of the vehicle measuring ranges up to 80 meters at 361 samples/sweep, 37.5 sweeps/second, with the samples interleaved 0.5 degrees apart. There were also four Sick LMS-291 ladars, similar in most respects except that they each swept a 90-degree arc while collecting 181 samples/sweep, 75 sweeps/second. The arrangement and function of the ladars is further discussed in Section 7.3. A monocular camera system from Mobileye, used for road

finding, was mounted at the "hood ornament" position, while a stereo system from Toshiba looked out through the windshield.

The On-Board Diagnostics Bus II (OBDII) provided limited access to vehicle data; it was mainly used to control the vehicle's horn.

7.2.2 Software

In order for multiple developers to work in parallel, it was plain we needed a system that would give us the freedom to write code in diverse places like deserts and stadium parking lots while still maintaining all the features of a revision control system. We found this in the peer-to-peer revision control system, `darcs` [Roundy, 2005], which maintains repositories containing both source code and a complete history of changes, and allows a developer to push or pull individual patches from one source repository to another. Using `darcs`, we achieved a very tight development-test cycle despite our diverse working environments.

The software ran on a Linux laptop and was divided into two main applications. The program which made decisions based on sensor input, controlled the actuators, and recorded events is known as `golem`. It had no connection to the visualization software `dashboard` other than through the log files it created, which were human-readable plain text. Besides being written to disk, these log files could be piped directly from `golem` to `dashboard` for realtime visualization or replayed offline by `dashboard` at a later time.

Commands could be typed directly to `golem` at the console or received from `dashboard` over a UDP port. The commands were designed to be simple and easily typed while driving in a moving vehicle on a dirt road. While this was convenient, it might have been even more convenient to be able to use an analog control or at least the arrow keys to adjust control parameters in real time.

7.2.2.1 Golem

The main software for data capture, planning, and control, `golem` consisted of two threads. The main thread was completely reactionary and expected to be realtime; it took in data from the sensors, processed it immediately, and sent commands to the actuators. The low level drivers, the state estimators, ladar and obstacle filters, the road/path follower, the velocity profiler, and the controllers all ran in this thread. A second planning thread was allowed to take more time to come up with globally sensible paths for the vehicle, based on snapshots of the accumulated sensor data received at the time the thread was initiated.

`Golem` could be driven by realtime sensor data, by a simple simulator, or from previously recorded log data. The simulator was invaluable for debugging the high level behaviors of the planner, but its models were not accurate enough to tune the low level controllers. The replay mode allowed us to debug the ladar obstacle filters and the state estimators in a repeatable way without having to drive the vehicle over and over.

7.2.2.2 Dashboard

`Dashboard` was created from scratch using C, Gtk+2, and OpenGL, to be an extensible visualization tool which would allow us to play back our log files and learn from their contents. A typical screenshot is shown in Figure 7.4.

The core of `dashboard` consists of a log file parsing library, an OpenGL representation of the world, and a replaying mechanism which controls the flow of time. `Dashboard` takes as input a log file from disk, standard input, or a UDP network connection, and uses rules described in a configuration file to convert human-readable log lines into an internal representation that is easy for other program components to index and access. Log lines are parsed as soon as they are available, by the Perl compatible regular expression (PCRE) library. The internal representation is displayed for the user through the OpenGL window. The user has the ability to play log files, pause them, rewind, fast-forward, measure and move around in the rendered world, and selectively view sensor visualizations they are interested in. This proved indispensable in our development effort.

The data is visualized in a number of different ways, as we found that no one representation was suitable for all debugging situations. The main section is a graphical representation of the world which is viewable from a 2d top-down point of view and also from a 3d point of view. It is rendered using OpenGL. There is also a section that displays inputs as textual data for precise readings. Yet another section contains a set of user programmable graphs. Because of the intentionally generic internal representation, adding a new element to the visualization, such as data from a new sensor, is a very simple process.

7.3 Ladar Obstacle Detection

We considered a non-traversable obstacle to be an object or ground feature which (a) represented a rapid apparent change in ground elevation, with slope magnitude greater than 30 degrees, or an actual discontinuity in ground surface; (b) presented a total change in ground height too large for the vehicle to simply roll over; (c) if discontinuous with the ground, was not high enough for the vehicle to pass under. This was an adequate definition of "obstacle" for the DARPA Grand Challenge.[1] Since obstacles result from changes of ground elevation, the most critical information comes from comparisons of surface measurements adjacent in space.

We did not use sensor data integrated over time to build a map of absolute ground elevation in the world frame, on the hypothesis that this is unnecessarily one step removed from the real information of interest. Instead, we tried to directly perceive, or infer from instantaneous sensor data, regions of rapid change

[1] Ideally one would like to classify some objects, such as plants, as "soft obstacles" that can be driven over even if they appear to have steep sides. Other hazards, such as water or marshy ground, cannot be classified simply as changes in ground elevation. But this increased level of sophistication was not necessary to complete the DGC and remains a topic of future work for us.

Fig. 7.4. `Dashboard` visualization of Golem 2 entering a tunnel during an NQE run

in ground elevation in the body-fixed frame, and then maintain a map of those regions in the world frame. We did not concern ourselves with any ground slope or surface roughness that did not rise to the threshold of making the ground non-traversable.

7.3.1 Ladar Geometry

We used Sick LMS-291 and LMS-221 laser scanners as our primary means of obstacle detection. It is interesting that the many DGC teams using two-dimensional ladars such as these found a wide variety of ways of arranging them. These ladars sweep a rangefinding laser beam through a sector of a plane, while the plane itself can be rotated or translated, either by vehicle motion or by actuating the ladar mount. The choice of plane, or more generally the choice of scan pattern for any beam-based sensor, represents a choice between scanning rapidly in the azimuthal direction, with slower or sparser sampling at different elevation angles, or the reverse.

A ladar scanning in the vertical plane has the significant advantage that the traversability of the scanned terrain is apparent from a single laser sweep. For example, it is easily determined from the single vertical-plane scan in Figure 7.5 that the ground is continuous, flat, and traversable from a few feet before the truck up to the point where there is a non-traversable vertical surface taller than the vehicle's wheels.

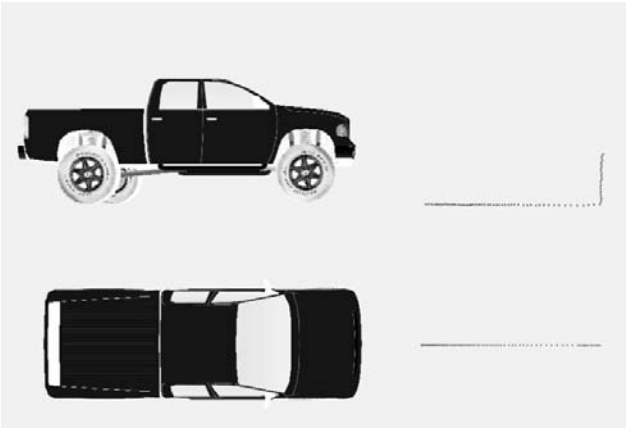

Fig. 7.5. Returns from a ladar oriented in a vertical plane

In our case, a single laser sweep takes 1/75 second and individual sample points are separated by 1/13575 second. On this time scale it is not likely that motion or vibration of the vehicle could distort the vertical scan sufficiently to alter this interpretation (make the traversable ground appear non-traversable or vice versa). Similarly, while small errors in the estimated pitch of the vehicle would induce small errors in the estimated location of the non-traversable obstacle, it is not likely that they could cause a major error or prevent the perception of an obstacle in the right approximate location. Against these advantages is the obvious drawback that a vertical slice only measures the terrain in a single narrow direction. Even if there are multiple vertical scanners and/or the vertical plane can be turned in different directions, the vehicle is likely to have a blinkered view with sparse azimuthal coverage of the terrain, and may miss narrow obstacles.

Conversely, a scan plane which is horizontal or nearly horizontal will provide good azimuthal coverage, and clearly show narrow obstacles such as fenceposts and pedestrians, but the interpretation of any single horizontal scan in isolation is problematic. Lacking measurements at adjacent elevation angles, one cannot determine if a return from a single horizontal scan is from a non-traversable steep surface or from a traversable gently sloping one. Therefore the information from multiple horizontal scans must be combined, but since the crucial comparisons are now between individual measurements taken at least 1/75 seconds apart instead of 1/13575 seconds apart, there is a greater likelihood that imperfectly estimated motion or vibration will distort the data. Small errors in pitch, roll, or altitude could cause the vehicle to misapprehend the height of a ground contour and lead to totally erroneous classification of terrain as traversable or non-traversable.

Our approach was to use a complementary arrangement of both vertical and nearly-horizontal ladars. On the Golem 2 vehicle, there are two nearly-horizontal ladars and three vertically-oriented ladars mounted on the front bumper. The idea is that the vertically-oriented ladars are used to form a profile of the general

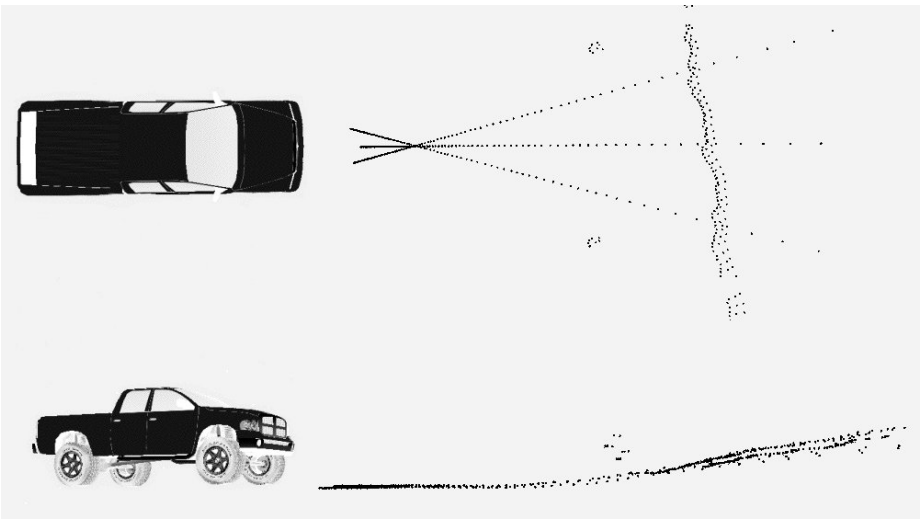

Fig. 7.6. Ladars distinguish obstacles from the ground surface

ground surface in front of the truck, in the truck body-fixed frame (as opposed to a world-fixed frame). The ground model we fit to the data was piecewise linear in the truck's direction of motion and piecewise constant in the sideways direction. The model interpolated the most recent available ladar data. In locations beyond the available data, the ground was assumed to have constant altitude in the body-fixed frame.

The apparent altitude of returns from the nearly-horizontal ladars relative to the ground model was then computed to see if those returns were (a) consistent with a traversable part of the ground model; (b) consistent with a non-traversable part of the ground model; (c) apparently from an elevation moderately higher than the notional ground; or (d) apparently from an object so far above the notional ground that the vehicle should be able to pass under it. In either case (b) or (c), the ladar return is classified as arising from a possible obstacle; after several of these returns are received from the same location at different vantage points, the presence of an obstacle is confirmed. In practice, an object such as a parked car will be represented as a cluster of adjacent obstacles.

For example, Figure 7.6 illustrates ladar data from a portion of the NQE course. The returns from the horizontally-oriented ladars indicated a long straight feature crossing in front of the vehicle, and two smaller features or clusters of returns. The three vertical ladars measure the ground profile in three directions and reveal that the truck is driving towards the foot of an upwards incline. The "long straight feature" is merely a surface contour of the upward-sloping ground, and therefore, since the incline is not too steep, should not be considered an obstacle. The two smaller clusters of returns, however, appear to be at a significant altitude above the notional ground, and are interpreted as non-traversable obstacles. In fact, these are traffic cones.

The entire process is $O(n)$ in the number of ladar measurements n, with a large array providing $O(1)$ access to accumulated obstacle detections in a given location. A many-to-one mapping between latitude-longitude coordinates and cells of the array is needed, with the properties that (a) each connected preimage of an array cell is at least approximately the same physical size and (b) any two distinct regions which are mapped to the same array cell are sufficiently far apart that they will not both need to be considered within the planning horizon of the vehicle. A simple method, which works everywhere except in the immediate vicinity of the poles, is to define a latitude-dependent cell size so that a different integral number of array cells correspond to a fixed extent of longitude at each latitude. It is acceptable that fewer array cells are needed further from the equator and therefore some array cells will have more preimage locations than others. The method could be adjusted to deal with the polar neighborhoods by exception.

Once confirmed, obstacles persist unless and until a period of time (e.g., one second) passes with no ladar returns detected from that location, in which case the obstacle expires. This enables the system to recover from false positive detections and also gives a rudimentary capability to deal with moving obstacles, since obstacles can disappear from one place and be re-detected somewhere else. However we did not attempt to form any velocity estimate of moving objects, and the robot generally operates on the assumption that other objects have zero velocity in the world frame.

In order to save computational time and memory, we did not wish to track obstacles which were well outside the DGC course boundaries. Off-course obstacles should be irrelevant to the vehicle's planning problem, and regularly checking to see whether they have expired should be a waste of time. Therefore non-traversable surfaces off the course were not classified as obstacles as long as the vehicle itself remained on the course. However, sensor measurements indicating off-course obstacles were allowed to accumulate, so that if the vehicle departed from the course boundaries for any reason, and was required to consider off-course obstacles as significant, these obstacles would pass the confirmation threshold very quickly.

7.3.2 Empirical Ladar Detection Results

The NQE obstacle course provided a uniform test environment with well-defined obstacles, of which many different sensor recordings presumably now exist, so it may be useful to report our ladar classification results during traversals of this obstacle course. The ranges at which the intended obstacles (parked cars, tire stacks, tank traps) were recognized as non-traversable by the vehicle are shown in Figure 7.7.

DARPA lined the boundaries of the course with traffic cones, which the vehicle, relying on its own GPS localization measurements, might consider cones to lie either inside or outside the course boundary. Those traffic cones considered to be on the course were classified as non-traversable at ranges indicated in Figure 7.8. Of the four outliers on the low end (cones which were not identified

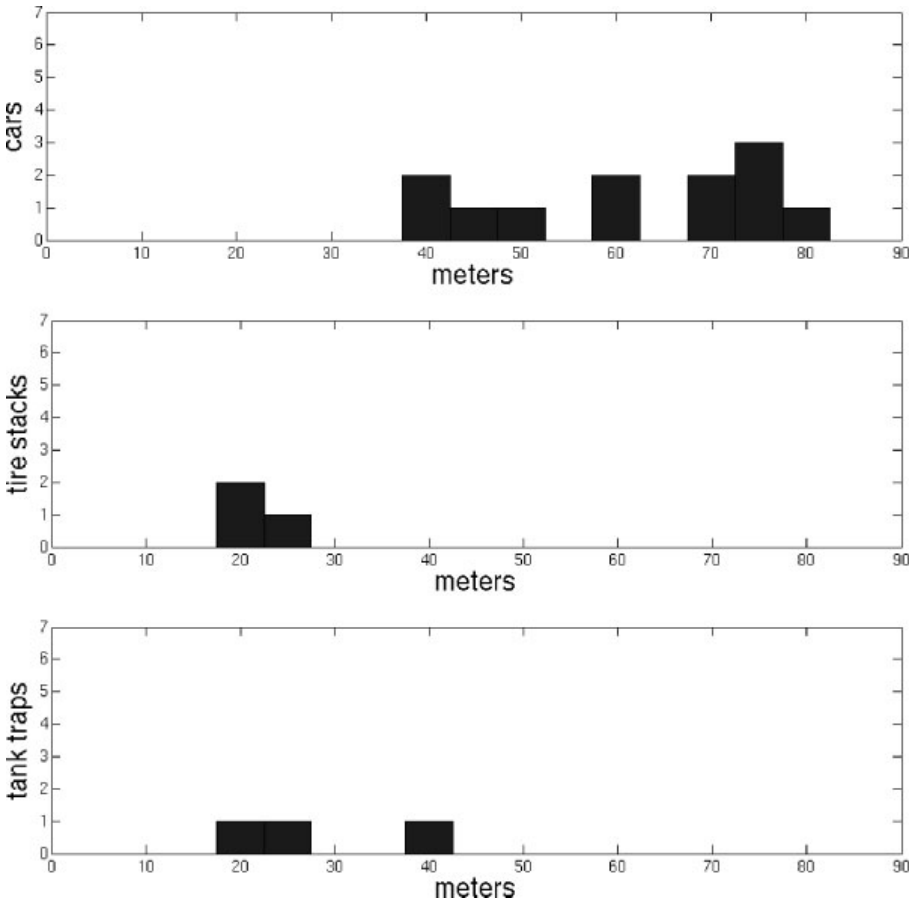

Fig. 7.7. Histograms of NQE obstacles arranged by range of first detection

until they were within less than twenty meters range), two cones were occluded from early view by other objects, and two were at the base of an incline and presumably could not be detected until the vehicle crested the top of the slope and headed down.

There were zero false negatives. Every intentional obstacle and cone on the course was correctly classified, and once classified, was persistently regarded as an obstacle as long as it remained within the 180-degree forward sensor arc of the vehicle.

There were, however, a considerable number of false positives, classifying terrain which was traversable as non-traversable. If these false obstacles appeared in the path of the vehicle, the vehicle would of course try to plan a new trajectory around them, and reduce speed if necessary. However, the false obstacles not did persist as the vehicle approached. Instead all false obstacles eventually expired,

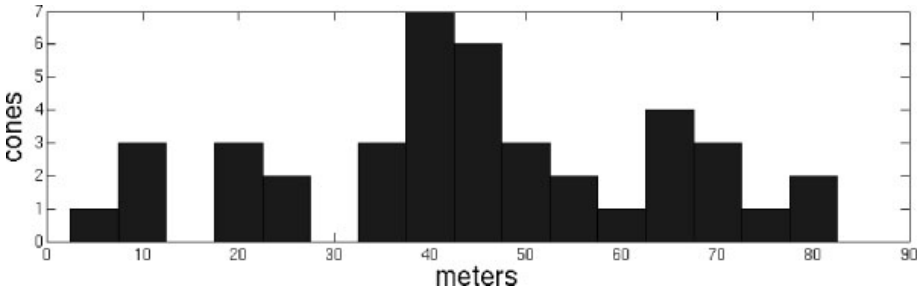

Fig. 7.8. Histogram of NQE traffic cones arranged by range of first detection

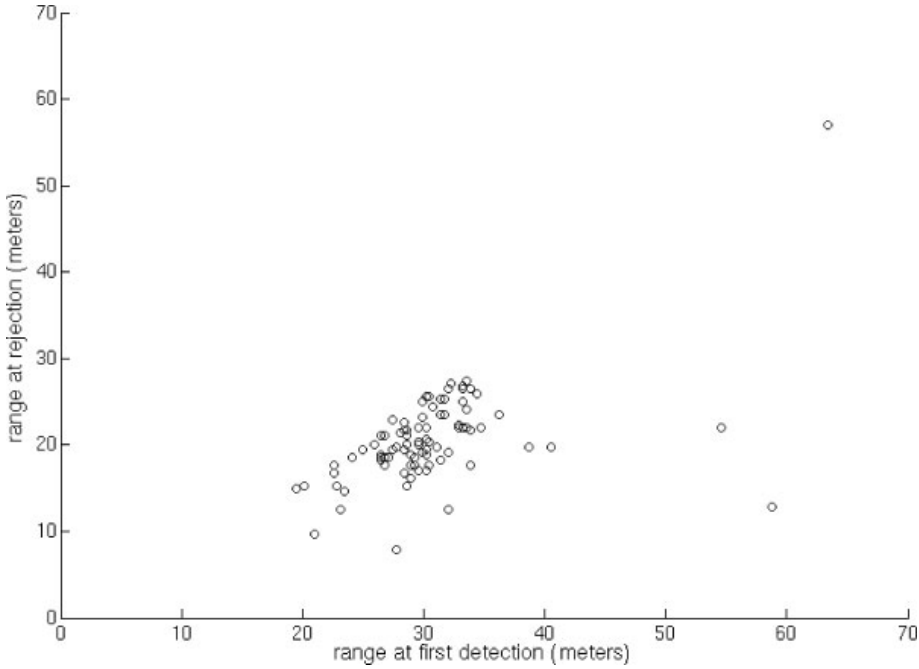

Fig. 7.9. False positive events in which obstacles were "detected" but subsequently rejected

leaving the vehicle free to drive over that terrain. The ranges at which false obstacle detections first appeared and then expired are shown in Figure 7.9. The mean range at first appearance was 30.5 meters; the mean range at expiration was 20.5 meters.

Some of the false positives coincided with irregularities in the ground surface or debris strewn on the course. In hindsight, it appears likely that a large percentage of the false positives were caused by reflective markings on the edges of the

speedway track, causing off-axis reflections of the laser beams and confusing the ladar range measurement. We infer this from the location of the false positives but have yet to examine the phenomenon through deliberate experiment. Similar problems occurred with reflective road markers during the GCE.

False positives very seldom occurred when the vehicle was travelling on a continuous, unmarked road surface, e.g., the asphalt track at the NQE, or a dirt road in the GCE. The main NQE performance impact of the false positives was to cause the vehicle to hesitate at transitions crossing the marked edge of the asphalt. However, the vehicle still managed to maintain a high average speed and complete the obstacle course in competitive times. (See Table 7.1.)

7.4 Avoider

We accomplished vehicle trajectory planning using a custom recursive algorithm described in this section. In keeping with the general philosophical approach of the team, we attempted to design for a minimal computation footprint, taking strategic advantage of heuristic rules and knowledge implicit in the DARPA-provided RDDF route. The method bears a resemblance to the generation of a probabilistic roadmap [Kavraki et al., 1996] or a rapidly-exploring random tree [Frazzoli et al., 2002, LaValle, 1998], in that we generate a graph of vehicle configurations connected by smooth trajectories, but instead of generating new configurations on the boundary of an expanding graph, we recursively generate trajectory midpoints in search of the best feasible path connecting a start configuration and end configuration.

In our search for high efficiency in practice, we forfeited any guarantee of the global optimality of our solution paths and even predictable convergence conditions for the routine. However, we found that in the vast majority of cases, traversable paths are found rapidly without burdening the processing resources of the vehicle.

The stage for the planning algorithm is a Cartesian two-dimensional map in the vicinity of the initial global coordinates of the vehicle, populated with a number of discrete obstacles. The obstacles are represented as point hazards that must be avoided by a particular radius or as line segments that can be crossed in only one direction. Point obstacles are centered on locations in the plane that have been identified as non-traversable by the ladar system. The buffer radius surrounding each point obstacle includes the physical width of the vehicle and additionally varies based on the distance from the vehicle to the hazard. A real object such as a wall or car is represented by a cluster of such points, each with its associated radius. Line segment obstacles arise from the Route Definition Data File (RDDF) which specifies corridor boundaries for the vehicle. The vehicle trajectory is a single curve describing the motion of the vehicle frame in the plane.

Our task is to find a drivable path from an initial configuration to a destination configuration that does not encroach on any obstacle.[2] The destination

[2] At this time, we have no classification of obstacles by importance. All obstacles are regarded as equally non-traversable and equally to be avoided.

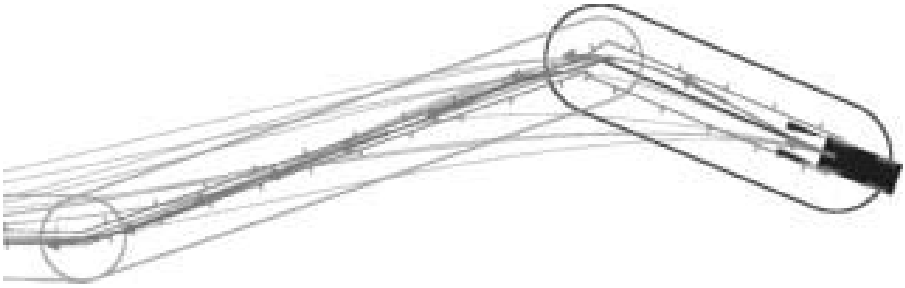

Fig. 7.10. Visualization of Avoider algorithm at GCE start line

is taken to be the center of the RDDF corridor at some distance ahead of our sensor horizon. The simplest and presumably most frequently occurring scenario will require the vehicle to simply drive forward from to without performing any evasive maneuvers. We represent this scenario with a directed acyclic graph containing two nodes and a single connecting edge. This choice of data structure later allows us to take advantage of a topographically sorted ordering that enables the least cost traversal of the graph to be found in linear time (ON edges). Each node in the graph is associated with a possible configuration of the vehicle and each edge is associated with a directed path connecting a starting configuration to a target configuration. At all times, care is taken to maintain the topological correspondence between the nodes in the graph and the locations in the local map.

After initialization we enter the recursive loop of the algorithm, which consists of three stages:

1. Graph Evaluation
2. Path Validation
3. Graph Expansion

This loop is executed repeatedly until a satisfactory path is found from to, or until a watchdog termination condition is met. Typically we would terminate the planner if a path was not found after several hundred milliseconds.

Figure 7.10, Figure 7.11, and Figure 7.12 offer a visualization of the path planning process. Nodes are indicated by bright purple arrows while considered trajectories are drawn as purple lines.

In Graph Evaluation, trajectory segments and cost factors are computed for all new edges in the graph. Each trajectory segment is calculated from the endpoint node configurations that it connects and is afterwards checked for possible obstacle collisions. There may be a large number of different trajectory segments to check and a large number of obstacles currently being tracked, bearing in mind that a single real object of any significant horizontal extent will be tracked as a cloud of smaller obstacles which are close together. In order to efficiently do collision checking, we use a wavefront propagation method to create a map

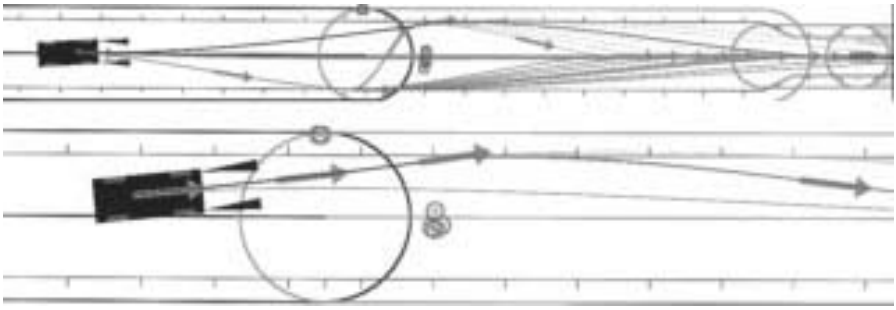

Fig. 7.11. Visualization of Avoider graph as the vehicle navigates around a tank trap obstacle during the NQE

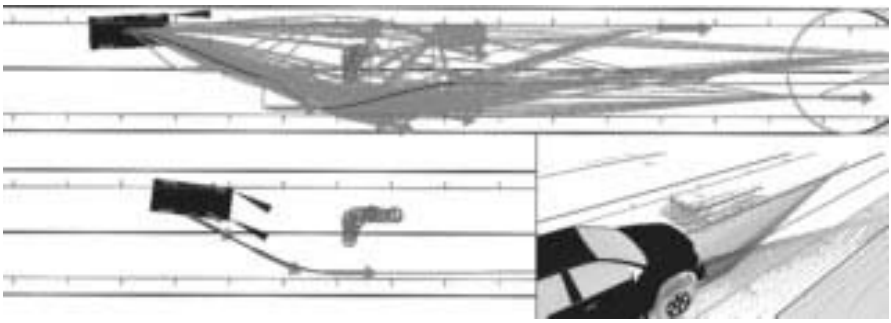

Fig. 7.12. Visualization of Avoider graph as the vehicle navigates around a parked car obstacle during the NQE

indicating approximate distance from each point to the nearest obstacle. Once this map is produced, trajectories can be rapidly checked against it for obstacle collisions. If no obstacles are intersected, the cost of traversal is generated from an integral of the instantaneous curvature along the trajectory. In our experience, the dynamic feasibility of a path can be adequately characterized by this single value.

The loop continues with Path Validation. The optimal path over the entire explored graph is found rapidly by reference to the topological sort of the nodes. If the cost is satisfactory the loop ends and a refinement procedure begins. This refinement process consists of a number of heuristic modifications that add nodes and edges to the graph to "smooth" calculated trajectories. As a final step, a velocity profile is computed along the solution, so that the vehicle's speed will be consistent with the curvature of the path and with any DARPA-imposed speed limits. The planning thread then updates the desired trajectory for the steering and velocity feedback control system.

If the total cost of the best graph traversal is too high, we enter the third phase of the algorithm: Graph Expansion. In this stage, edges that intersect obstacles are split at the point where a collision would occur. A number of new nodes are added to the graph for configurations selected according to heuristic avoidance conditions. Unevaluated edges connect these new nodes to the endpoints of the offending trajectory segment. Trajectory segments for these new edges are then computed when the recursive loop begins again with Graph Evaluation.

In the heuristic techniques applied to select good target configurations for obstacle avoidance, the position component of the target configuration is typically projected perpendicularly out from the obstacle or else interpolated between the obstacle position and the RDDF boundary. We found that vehicle heading was similarly best specified as a function of both the heading at the collision point and the RDDF-defined corridor direction.

7.5 Vehicle State Estimation

The path planning and control subsystems of Golem 2 needed a good estimate of the latitude, longitude, heading and the velocity of the vehicle, at a level of accuracy that was beyond that provided by on-board sensors like the C-MIGITS. Two different state estimators were implemented to carry out this task, as a means to provide analytic redundancy. The first state estimator used a model analogous to a bicycle for the vehicle and relied heavily on the history of state. The second estimator was based on a six-degrees-of-freedom rigid-body model, with the addition of a "soft" non-holonomic constraint on the vehicle's velocity enforced as a virtual heading measurement.

7.5.1 The Bicycle Estimator

In this section we will describe the working of the first estimator, henceforward referred to as the bicycle estimator. The bicycle estimator was a discrete time extended Kalman filter [Gelb, 1974, Kalman, 1960, Kalman and Bucy, 1961, Welch and Bishop, 1995], with the following inputs:

1. Latitude and longitude from a NovAtel GPS sensor at 20 Hz.
2. Rear axle velocity at 30 Hz from a custom Hall sensor system. A set of 16 magnets was installed on the rear axle. Two detectors were attached to the vehicle frame and passed a voltage pulse every time one of the magnets went past them. The two sensors enabled us to have a quadrature encoder, i.e., we were able to distinguish between forward and reverse motion. A discrete time two state Kalman filter used the voltage pulses as input and estimated the rate of rotation, which in turn was scaled by gear ratio and wheel radius to infer the velocity of the vehicle.
3. Steering angle from an absolute encoder at 20 Hz.

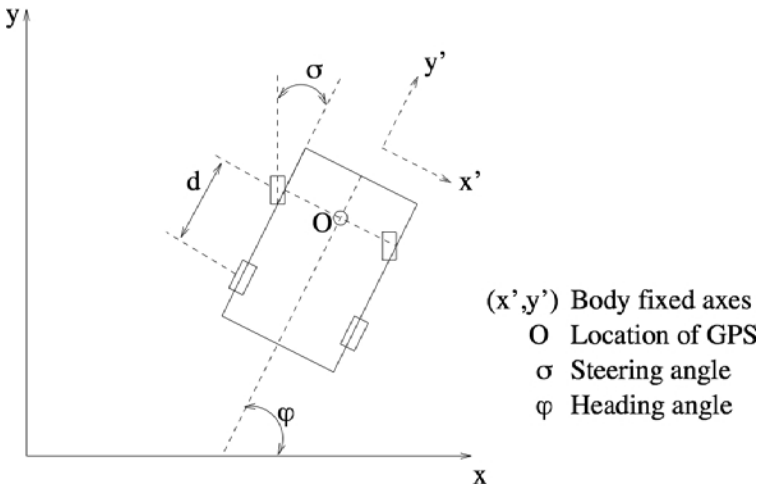

Fig. 7.13. The "bicycle" model

Rear axle velocity and steering encoder were used as inputs to the bicycle estimator. The state propagation equations for the bicycle estimator were:

$$\begin{aligned}
\tilde{x}_{k+1} &= \hat{x}_k + \hat{v}_k \Delta t \frac{\cos(\hat{\phi}_k + \hat{\sigma}_k)}{\cos \hat{\sigma}_k} \\
\tilde{y}_{k+1} &= \hat{y}_k + \hat{v}_k \Delta t \frac{\sin(\hat{\phi}_k + \hat{\sigma}_k)}{\cos \hat{\sigma}_k} \\
\tilde{\phi}_{k+1} &= \hat{\phi}_k + \frac{\hat{v}_k \Delta t}{d} \tan \hat{\sigma}_k
\end{aligned} \tag{7.1}$$

where as described in Figure 7.13, (x, y) are the local cartesian coordinates of the center of the front axle of the car. The GPS sensor is located at this point. The angle ϕ is the heading of the car with respect to the local x axis. The angle σ is the steering angle of the car as shown in the figure. v is the rear axle velocity of the car. The quantity d is the distance between the front and rear axle of the vehicle. A caret over a variable implies that the variable is an estimate, and a tilde over a variable implies that the variable is predicted or propagated forward. The time elapsed since the last state update is Δt. The subscripts in the above equations refer to successive time indices in state propagation.

The bicycle model worked well when the steering angle was small. For large steering angles however, the model is inaccurate and leads to considerable lag between the updated state and the GPS measurement. The model was also found to be inaccurate for high velocities, when the vehicle slips at turns. Thus, the steering angle, σ, was calculated as:

$$\hat{\sigma}_k = \hat{\gamma}_k (\sigma_{\text{measured}} - \hat{\sigma}_{\text{bias } k}) \tag{7.2}$$

where γ is the steering calibration factor which tries to compensate for model inaccuracies. The steering bias estimate σ_{bias} is used to compensate for bias in the measured steering angle. The rear axle velocity, v, was calculated as:

$$\hat{v}_k = \hat{S}_k(v_{\text{measured}}) \tag{7.3}$$

where v_{measured} is the velocity being input to the bicycle estimator. The slip factor S compensates for variations in the vehicle's apparent wheel radius, and for the fact that the vehicle may be going uphill or downhill, and tries to estimate slip. However, the slip factor was not able to track consistently long periods of slipping.

The state variables were latitude, longitude (translated to local Cartesian coordinates x,y), heading ϕ, slip factor S, and either steering bias σ_{bias} or steering calibration factor γ.

The steering calibration factor γ and the steering bias cannot be estimated simultaneously as the state variables become unobservable. So, the bicycle estimator was operated in the following 2 modes:

1. When the vehicle was going straight, $|\sigma_{\text{measured}}| < 3^\circ$, σ_{bias} was estimated.
2. When the vehicle was turning, $|\sigma_{measured}| > 3^\circ$, γ was estimated. This enabled compensating for model inaccuracies for hard turns.

7.5.2 The Six-Degrees-of-Freedom Estimator

In this section, we describe the design of the six-degree-of-freedom (6DOF) estimator. Like the bicycle estimator discussed previously, the 6DOF estimator is implemented as a discrete-time Extended Kalman Filter. The estimator is designed using fairly standard techniques for strap-down inertial navigation systems. Since a detailed model of the vehicle's dynamics is not available, the filter relies mainly on the rigid-body kinematic equations. However, due to the absence of a magnetometer or other means to measure the vehicle's orientation, and the need to be able to ensure convergence of the non-linear filter without requiring a initial calibration procedures, knowledge of the vehicle's dynamics is exploited in the form of virtual heading measurements from inertial velocity data.

The vehicle is modeled as a rigid body moving in the three-dimensional space; the state of the vehicle can hence be described by a position vector $p \in \mathbb{R}^3$, representing the location with respect to an Earth-fixed reference frame of the on-board Inertial Measurement Unit (IMU), a velocity vector $v = dp/dt$, and a rotation matrix $R \in SO(3)$, where $SO(3)$ is known as the Special Orthogonal group in the three-dimensional space, and includes all orthogonal 3 by 3 matrices with determinant equal to $+1$. The columns of R can be thought of as expressing the coordinates of an orthogonal triad rigidly attached to the vehicle's body (body axes).

The inputs to the estimator are the acceleration and angular rate measurements from the IMU, provided at 100 Hz, and the GPS data, provided at about 20 Hz. In addition, the estimator has access to the velocity data from the Hall sensors mounted on the wheels, and to the steering angle measurement.

In the following, we will indicate the acceleration measurements with $z_{\text{a}} \in \mathbb{R}^3$, and the angular rate measurements with $z_{\text{g}} \in \mathbb{R}^3$. Moreover, we will indicate with

Z_a the unique skew-symmetric matrix such that $Z_\mathrm{a} v = z_\mathrm{a} \times v$, for all $v \in \mathbb{R}^3$. A similar convention will be used for z_g and other three-dimensional vectors throughout this section.

The IMU accelerometers measure the vehicle's inertial acceleration, measured in body-fixed coordinates, minus the gravity acceleration; in other words,

$$z_\mathrm{a} = R^T(a - g) + n_\mathrm{a},$$

where a and g are, respectively, the vehicle's acceleration, and gravity acceleration in the inertial frame, and n_a is an additive, white Gaussian measurement noise. Since the C-MIGITS IMU estimates accelerometer biases, and outputs corrected measurements, we consider n_a as a zero-mean noise.

The IMU solid-state rate sensors measure the vehicle's angular velocity, in body axes. In other words,

$$z_\mathrm{g} = \omega + n_\mathrm{g},$$

where ω is the vehicle's angular velocity (in body axes), and n_g is an additive, white Gaussian measurement noise. As in the case of acceleration measurements, n_g is assumed to be unbiased.

The kinematics of the vehicle are described by the equations

$$\begin{aligned} \dot{p} &= v, \\ \dot{v} &= a, \\ \dot{R} &= R\Omega, \end{aligned} \tag{7.4}$$

in which Ω is the skew-symmetric matrix corresponding to the angular velocity ω, and we ignored the Coriolis terms for simplicity. As a matter of fact, this is justified in our application due to the low speed, relatively short range, and to the fact that errors induced by vibrations and irregularities in the terrain are dominant with respect to the errors induced by ignoring the Coriolis acceleration terms.

We propagate the estimate of the state of the vehicle using the following continuous-time model, in which the hat indicates estimates:

$$\begin{aligned} \dot{\hat{p}} &= \hat{v}, \\ \dot{\hat{v}} &= \hat{R} z_\mathrm{a} + g, \\ \dot{\hat{R}} &= \hat{R} Z_g. \end{aligned} \tag{7.5}$$

An exact time discretization of the above, under the assumption that the (inertial) acceleration and angular velocity are constant during the sampling time is:

$$\begin{aligned} p^+ &= p + v\Delta t + \tfrac{1}{2}\left(\hat{R} z_\mathrm{a} + g\right)\Delta t^2, \\ v^+ &= v + \left(\hat{R} z_\mathrm{a} + g\right)\Delta t, \\ R^+ &= R \exp(Z_g \Delta t). \end{aligned} \tag{7.6}$$

The matrix exponential appearing in the attitude propagation equation can be computed using Rodrigues' formula. Given a skew-symmetric 3 by 3 matrix M,

write it as the product $M = \Omega\theta$, such that Ω is the skew-symmetric matrix corresponding to a unit vector ω; then

$$\exp(M) = \exp(\Omega\theta) = I + \Omega \sin\theta + \Omega^2(1 - \cos\theta). \tag{7.7}$$

The error in the state estimate is modeled as a 9-dimensional vector $\delta x = (\delta p, \delta v, \delta\phi)$, where

$$\begin{aligned} p &= \hat{p} + \delta p, \\ v &= \hat{v} + \delta v, \\ R &= \hat{R}\exp(\delta\Phi). \end{aligned} \tag{7.8}$$

Note that the components of the vector $\delta\phi$ can be understood as the elementary rotation angles about the body-fixed axes that make $\hat{R}$ coincide with R; such a rotation, representing the attitude error, can also be written as $\delta R = \exp(\delta\Phi) = \hat{R}^T R$.

The linearized error dynamics are written as follows:

$$\frac{d}{dt}\delta x = A\delta x + Fn, \tag{7.9}$$

where

$$A := \begin{bmatrix} 0 & I & 0 \\ 0 & 0 & -RZ_{\mathrm{a}} \\ 0 & 0 & -Z_{\mathrm{g}} \end{bmatrix}, \qquad F := \begin{bmatrix} 0 & 0 \\ R & 0 \\ 0 & I \end{bmatrix}. \tag{7.10}$$

When no GPS information is available, the estimation error covariance matrix $P := \mathrm{E}[\delta x(\delta x)^T]$ is propagated through numerical integration of the ODE

$$\frac{d}{dt}P = AP + PA^T + FQF^T.$$

Position data from the GPS are used to update the error covariance matrix and the state estimate. The measurement equation is simply $z_{\mathrm{gps}} = p + n_{\mathrm{gps}}$. In order to avoid numerical instability, we use the UD-factorization method described in Rogers [2003] to update the error covariance matrix and to compute the filter gain K.

Since the vehicle's heading is not observable solely from GPS data, and we wished to reduce calibration and initialization procedure to a minimum (e.g., to allow for seamless resets of the filter during the race), we impose a soft constraint on the heading through a virtual measurement of the inertial velocity, of the form:

$$z_{\mathrm{nhc}} = \arctan\left(\frac{v_{\mathrm{East}}}{v_{\mathrm{North}}}\right) - \lambda\sigma,$$

where σ is the measured steering angle, and λ is a factor accounting for the fact that the origin of the body reference frame is not on the steering axle.

In other words, we effectively impose a non-holonomic constraint on the motion of the vehicle through a limited-sideslip assumption. This assumption is usually satisfied when the vehicle is traveling straight, but may not be satisfied

during turns; moreover, when the vehicle is moving very slowly, the direction of the velocity is difficult to estimate, as the magnitude of the velocity vector is dominated by the estimation error. Hence, the virtual measurement is applied only when the steering angle is less than 10 degrees, and the vehicle's speed is at least 2.5 mph (both these values were determined empirically).

The filter described in this section performed satisfactorily in our tests and during the DGC race, providing the on-board control algorithms with position and heading estimates that were nominally within 10cm and 0.1° respectively.

7.5.3 Modeling System Noise

The state was propagated and updated every time a GPS signal arrived. There was no noise associated with the state propagation equations (7.1). It was assumed that there was additive white gaussian noise associated with the inputs (v_{measured} and σ_{measured} for the bicycle model, and z_{a} and z_{g} for the six-degrees-of-freedom model). To efficiently track the constants in the state variables (S, σ_{bias}, γ), it was assumed that there was an additive white gaussian noise term in their propagation. It was also assumed that the GPS measurement (latitude and longitude) had some noise associated with it. The variances assigned to the noise processes described above were tuned to ensure that the innovations[3] were uncorrelated, and that the estimator was stable, and converged reasonably quickly. By tuning the variances, we can alter the "trust" associated with the history of the vehicle state or with the GPS measurement.

7.5.3.1 Determining the Appropriate GPS Measurement Noise Variance

The received data from the GPS consisted of the latitude, longitude and an HDOP (horizontal dilution of precision) value. HDOP is a figure of merit for the GPS measurement which is directly related to the number of satellites visible to the GPS antenna. The HDOP value is related to the noise in each measurement. However, our attempts at mapping the HDOP values to actual variances were futile as we did not observe a monotonic relationship. It was noticed that an HDOP of more than 5 usually corresponded to multipath reception in GPS and consequently, the GPS had an abnormally huge variance in that case. In the absence of any ad hoc relationship between the HDOP and noise variance, we opted to keep a constant variance associated with GPS noise if HDOP was less than 5 and a huge variance otherwise. Also, we noticed that the GPS noise was highly correlated and not white.

7.5.3.2 Projection of the Innovations

The bicycle estimator is based on a very intuitive model of the vehicle, which motivates us to consider the GPS innovations in a physically relevant reference

[3] Innovation of a measured data is the difference between the estimated and the measured data.

frame rather than any arbitrary reference frame. It is insightful to project the innovations into the local frame of the vehicle: parallel to the heading direction and perpendicular to it. The parallel and perpendicular body fixed axes are indicated in Figure 7.13.

For an ideal estimator, the innovations will be uncorrelated with each other. However, we tuned the estimator to just achieve innovations with zero mean. While tuning the estimator it was very useful to consider the parallel and perpendicular innovations. For example, a DC bias in the parallel innovations implied that we were not tracking the slip factor (S) adequately. Thus, to ensure a zero mean parallel innovation the variance associated with the propagation of slip factor should be increased.

7.5.3.3 Adaptive Shaping of the Innovations

The noise in the GPS data was highly correlated and there was very little a priori knowledge of the variance. Very often, especially when the vehicle would drive near a wall, or approach a tunnel, there would be highly erratic jumps in the GPS measurements due to multipath reflections. Without any a priori knowledge of the variance in such cases, the state estimate would bounce around a lot. This was undesirable as it would hamper the path planner and the obstacle detection subroutines in the main program.

To counter these "unphysical" jumps, once the estimator was converged, the innovations were clipped up to a certain maximum absolute value. For example, a GPS measurement corresponding to a perpendicular innovation of 2 meters in 0.05 sec while going straight is unphysical and so perpendicular innovation should be clipped to a nominal value (in our case, six inches). This prevented large jumps in state estimate, but had a grave disadvantage. It was observed that if the innovations were clipped to a fixed range, then in certain situations, the state estimate will lag far behind from "good" set of GPS measurements and take a long time to converge back. To prevent this from happening, the clipping limit of the innovations was determined adaptively as the minimum of either a fixed limit, or the mean of the innovations in the last two seconds scaled by a numerical factor slightly greater than unity. The parallel and perpendicular components of the innovation were clipped separately with different numerical constants used.

7.5.3.4 Countering the Time Delays

There was some finite delay between the time sensor data appeared on the bus and the time it was processed by the control program. Usually this delay was nominal ($\sim$50-200 μs), but it was sporadically very large. A large delay in GPS data manifested in large negative parallel innovation. However, the large innovation was clipped effectively by the method described earlier, and consequently did not effect the state estimate. In future, we plan to implement a routine which synchronizes the time between all the serial/USB data sources.

Performance of the state estimator while going under a bridge is shown in Figure 7.14. "*" represents the GPS data received in a local cartesian cordinate

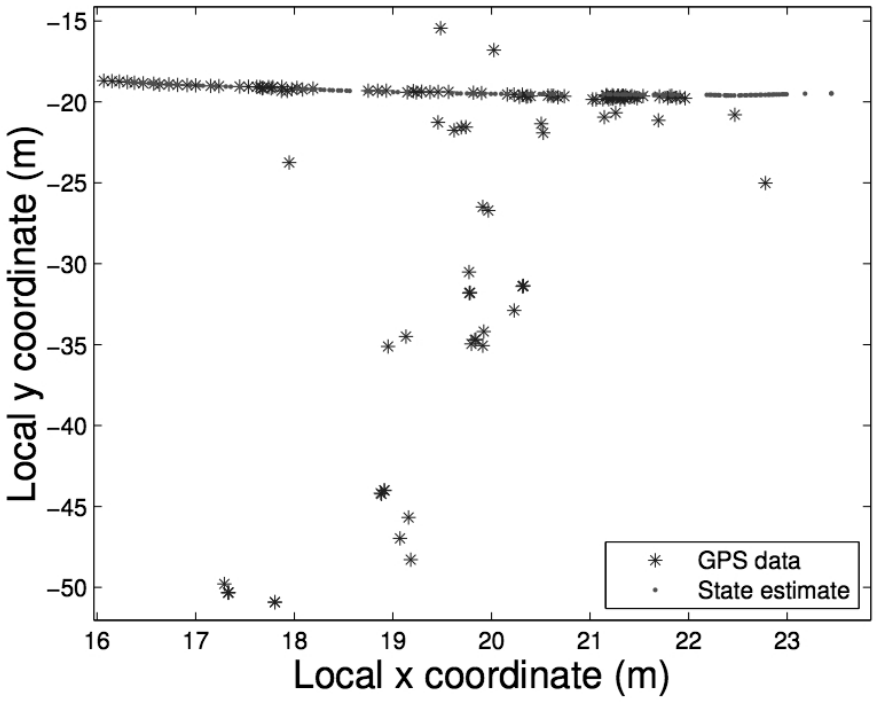

Fig. 7.14. Performance of state estimator while going under a bridge

system. "·" represents the estimated location of the truck. Golem2 was moving from left to right in this figure. Note that the GPS data has a huge variance due to multipath. Also note that the GPS noise is highly correlated, in this case the variance is huge in the direction perpendicular to the motion as compared to the parallel direction. As seen from the figure, the performance of the state estimator is immune to the large "unphysical" jumps in GPS data. Ocassional jumps of length $\sim 0.3m$ are observed in the successive updates of estimated state. These correspond to a delay in received GPS data and large negative parallel innovations as discussed in Section 7.5.3.4.

7.5.3.5 Special Modes of Operation

A couple of situations required special handling instructions:

- GPS signal was observed to drift considerably when the vehicle was at rest. Not only was this unphysical, but it is also detrimental to path planning. Thus when the vehicle was going considerably slowly or was at rest, the variance assigned to the GPS measurements was increased significantly. In such a case, the bicycle estimator essentially worked like an integrator rather than a filter.

- It was observed that the GPS signal would occasionally jump discretely. These jumps usually corresponded to the presence of a power transmission line nearby. The difficulty was that the GPS took a while (>2 s) to re-converge after the jumps. These unphysical jumps were easily detected from the jumps in the parallel and perpendicular innovations. After the detection of such jumps, the GPS variance was increased until it was trustworthy again, i.e., until the innovations were within a certain limit again.

7.5.3.6 Initialization of the IMU

One advantage of this model was that it converged very fast while going straight, usually in approximately 5 s. Once the bicycle model converged, the heading estimate was used to initialize the IMU. While going straight, the performance of the bicycle estimator was extremely good as the heading estimate was within 0.5° of the IMU computed heading. However, on sharp turns, the heading estimate was up to ∼3° from the IMU computed heading. Thus, IMU computed heading was given more importance especially when GPS signals were bad. In future, we plan to extend the bicycle model to include the angular rotation and linear displacement data from the IMU.

7.6 Vision Systems

For computer vision, desert paths present a different challenge from paved roads, as the environment is far less structured, and less prior information is available to exploit in constructing algorithms. Our approach was to integrate existing vision technologies which have been proven to work on-road but have the potential to be transferred to the off-road domain. These include learning-based road-finding and binocular stereo reconstruction.

7.6.1 Mobileye Vision System

Golem 2 was equipped with a sophisticated vision system, created by Mobileye Vision Technologies Ltd., consisting of a single camera and a dedicated processing unit. On-road, the Mobileye system can find lane boundaries and detect other vehicles and their positions in real-time. The system was adapted to the off-road environment by Mobileye and the Hebrew University of Jerusalem.

The Mobileye system combines region-based and boundary-based approaches to find path position and orientation relative to the vehicle. The two approaches complement each other, thus allowing reliable path detection under a wide range of circumstances. Specifically, we use a variety of texture filters together with a learning-by-examples Adaboost [Freund and Schapire, 1996] classification engine to form an initial image segmentation into path and non-path image blocks. In parallel, we use the same filters to define candidate texture boundaries and a projection-warp search over the space of possible pitch and yaw parameters in order to select a pair of boundary lines that are consistent with the texture

gradients and the geometric model. Both the area-based and boundary-based models are then combined (weighted by their confidence values) to form a final path model for each frame.

7.6.1.1 Region-Based Path Detection

The gray-level image is divided into partially overlapping blocks. A filter bank is applied to all image pixels and a descriptor vector is generated per block. The descriptor contains the mean and standard deviation of the filter response over the block for each of the 16 filters. Each entry in the texture descriptor can be considered a "weak" learner in the sense that it forms class discrimination. The iterative Adaboost algorithm combines the weak learners to form a powerful classification engine that assigns a *path* or *non-path* label to every block according to the training data. The training data was extracted from 200 images that were gathered on various parts of the 2004 Grand Challenge route. The block classification by itself is not sufficient for autonomous vehicle path planning because about 10 percent of the blocks are expected to be misclassified. Filtering methods are used to clear some of the misclassified blocks, followed by detection of path boundaries. The path boundaries are derived via a minimal error separating line on each side of the path. The system confidence is calculated from the separation quality and the sensing range (the distance to the farthest point that we can reliably identify as path). Figure 7.15 shows the results of each of the main parts of the algorithm.

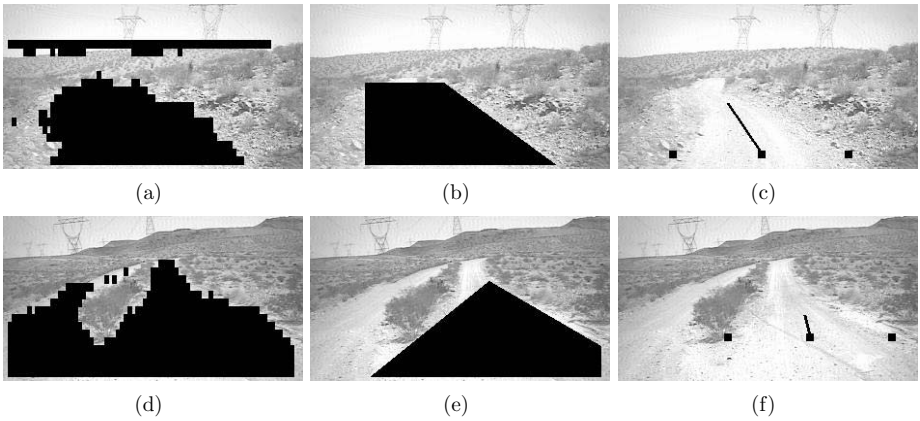

Fig. 7.15. In (a) and (d) blocks are classified by texture into *path* or *non-path*. In (b) and (e), sky blocks are removed and three lines, arranged in a trapezoid, are fitted to the *path* blocks. The trapezoid is considered to represent the path. Finally, in (c) and (f) the path boundaries are calculated at a given distance ahead of the vehicle (5 meters in this example) together with the path center and heading angle.

7.6.1.2 Boundary-Based Path Detection

The boundary-based technique does not rely on prior learned texture information. Instead, it makes the assumption that the path texture properties are different than the surrounding non-drivable areas. For this cue to be reliable, we have to constrain the solution to a strict geometric model where the path boundaries lie on straight parallel edges. This allows us to reduce the problem of finding a drivable path to 4 degrees of freedom: (x, y) position of the vanishing point, and left and right distance to the edge of the path. The geometric constraints resulting from assuming a flat world, perspective camera, and parallel path boundaries in the world suggest the following projection-warp scheme per frame: given a hypothesis of pitch and yaw angles of the camera, the image is warped to form a top view in world coordinates. In the warped image the path boundaries are supposed to be *parallel* vertical lines if indeed the pitch and yaw angles are correct. A projection of the image texture edges onto the horizontal axis will produce a 1D profile whose peaks correspond to vertical texture edges in the warped image. We look for a pair of dominant peaks in the 1D profile and generate a score value which is then maximized by search over the pitch and yaw angles via iterating the projection-warp procedure just described. The search starts with the pitch and yaw angle estimates of the previous frame followed by an incremental pitch and yaw estimation using optic-flow and a small motion model:

$$xw_x + yw_y = yu - xv \tag{7.11}$$

where (u, v) are the flow (displacements) of the point (x, y) and w_x, w_y are the pitch and yaw angles. The warped image is divided into overlapping 10×10 blocks with each pixel forming a block center. Using the same filter bank as in the

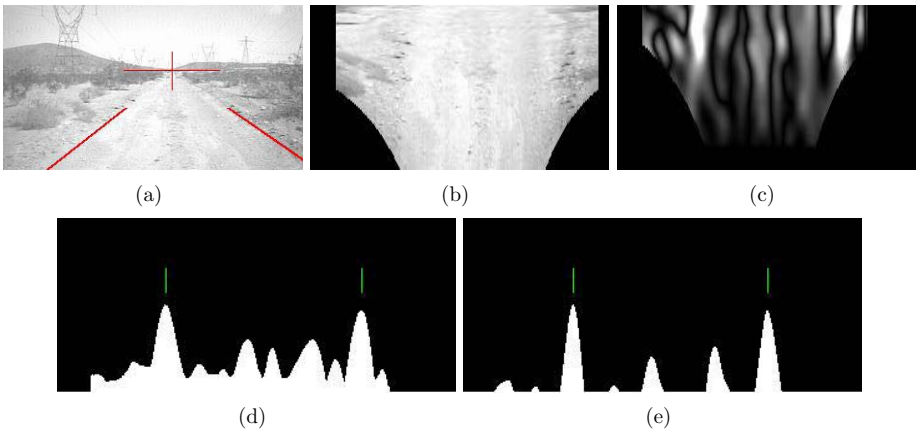

Fig. 7.16. Projection-Warp Search: (a) original image with the overlaid path boundary and FOE results. (b) the warped image. (c) texture gradients magnitude. (d) projection: vertical sum of gradients. (e) projection profile followed by convolution with a box filter. The two lines on top of the histogram marks the path boundaries.

region-based method, we estimate the the likelihood $e^{-\Delta}$ that the vertical line passing through the block center forms a texture gradient where Δ is the L_1 distance between the texture vector descriptors of the two respective halves of the block. To check a hypothesis (for pitch and yaw), we project the horizontal texture gradients vertically onto the x-axis and look for peaks in this projection. An example result of this projection is shown in Figure 7.16(d). The path boundaries and other vertical elements in the image create high areas in the projection, while low areas are most likely caused by vertical texture gradients that are not continuous and created by bushes, rocks, etc. The peaks in this projection are maximized when the vanishing point hypothesis is correct and the path edges (and possibly other parallel features) line up. By finding the highest peaks for these hypothesis, our system is able to find the lateral position of the left and right boundaries. Figure 7.16(e) shows the "cleaned up" 1D projection profile and the associated pair of peaks corresponding to the path boundary lines.

7.6.1.3 Performance

The system was implemented on a Power-PC PPC7467 1GHZ running at 20 frames per second. The camera was mounted on a pole connected to the front

Fig. 7.17. The camera is mounted inside a housing atop a pole connected to the front bumper. This allows a better field of view than mounting the camera on the windshield.

bumper (Figure 7.17) to allow maximal field of view. We tried both 45-degree and 80-degree field of view lenses, and found the latter to be more suitable for autonomous driving where the vehicle is not necessarily centered over the path. For our applications, the most meaningful overall system performance measure is to count how often (what fraction of frames) the system produced correct path edge positions and, where appropriate, heading angles. Furthermore, it is crucial for the system to know when it cannot determine the path accurately, so that the vehicle can slow down and rely more on information from the other sensors. Our results are broken up by different terrain types. For each, representative challenging clips of 1000 frames were selected and the system performance scored on these sequences by a human observer. The path edge distance accuracy was computed by observing the position of the road edge marks approximately 6 meters in front of the vehicle. A frame was labeled incorrect if the path edge marker at that location appeared to be more then 30 cm ($\approx$ 18 pixels) away from the actual path boundary. For straight paths the perceived vanishing point of the path was also marked, and our algorithm's heading indicator was compared to the lateral position of this point.

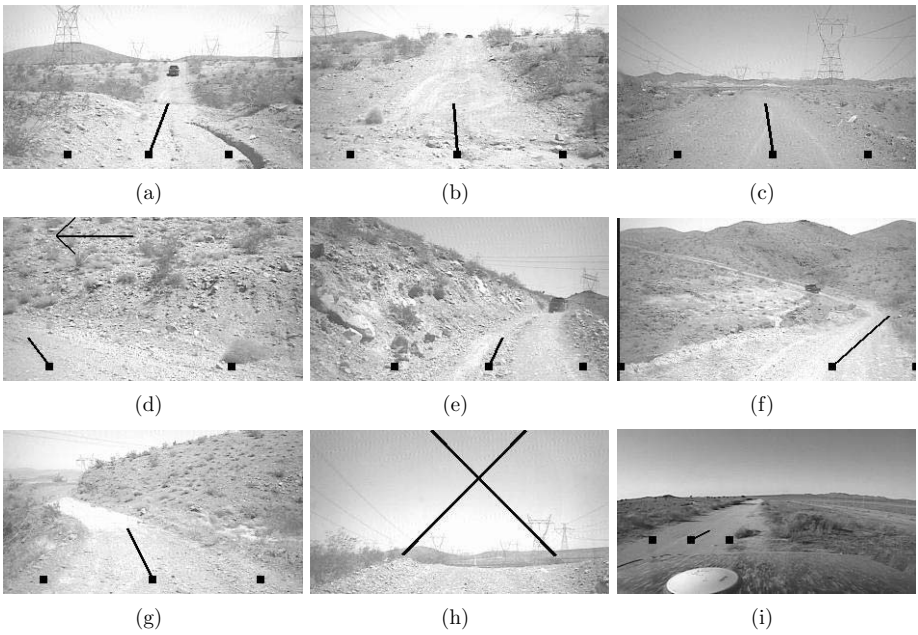

Fig. 7.18. Sample images and system output from 6 hours of driving in the Mojave desert. The path is marked by two left-right boundary points and a center point with heading orientation. The "X" mark in (h) coincides with zero confidence due to short range of visible path. In (i) the path is detected even though the vehicle is not centered on the path (a common situation in autonomous driving).

On relatively straight segments with a comfortably wide path, our system reported availability (high system confidence) **100%** of the time while producing accurate path boundary locations **99.5%** of the time. The mean angular deviation of the heading angle from the human marked vanishing point was 1.7 deg.

The second test clip is an example of more uneven terrains with elevation changes. Here the vehicle passes through a dry river ditch (Figure 7.18(b)) where both the path texture and scene geometry is difficult. When our vehicle is reaching the crest of the hill (Figure 7.18(h)) only a short segment of road is visible. In this case, the system reported unavailability (low confidence) **8%** of the time. When available, however, the accuracy in boundary locations was **98%**.

The final clip contains a winding mountain pass (Figure 7.18(g)), difficult due to path curvature as well as texture variation. Despite these, our system was available throughout the clip and achieved an accuracy of **96%** in detecting the path boundary.

7.6.1.4 Integration

The combination of the learning and geometric approaches yields high quality results with confidence estimates suitable for integration into our control systems. Output from the Mobileye system – which included path boundaries, orientations, and pitch – was integrated into the Golem path planner. Path boundaries were marked as line obstacles, projected onto the world coordinate frame using the known transformation between the Mobileye camera, the GPS antenna, and the road surface. Figure 7.19 shows the detected trail boundary and trail center relative to the vehicle and to the vehicle's actual trajectory.

7.6.2 Toshiba Stereo Vision System

The Golem/UCLA Team experimented with a variety of stereo vision systems for the purposes of obstacle detection, ground-plane registration, and path finding. Fixed-camera stereo is a well studied problem, and there are a number of academic and commercial systems of varying quality. Long range sensing is essential for use in a high-speed automobile since time is of the essence; it is impractical and sometimes impossible to stop to analyze nearby obstacles. This requirement translates into a wide baseline separating the stereo cameras, since the maximum range of the system is determined by this distance. A further performance constraint is processing time, since latency can introduce dangerous errors in path planning and control. Toshiba Research in Kawasaki, Japan, developed a stereo system for on-road driving at high speeds, which we installed on Golem 2. Due to insufficient time, we did not integrate the stereo system into the control system of the vehicle before the GCE, but there is no doubt that it has a high potential for successful autonomous driving. In this section we describe the hardware configuration and implementation details.

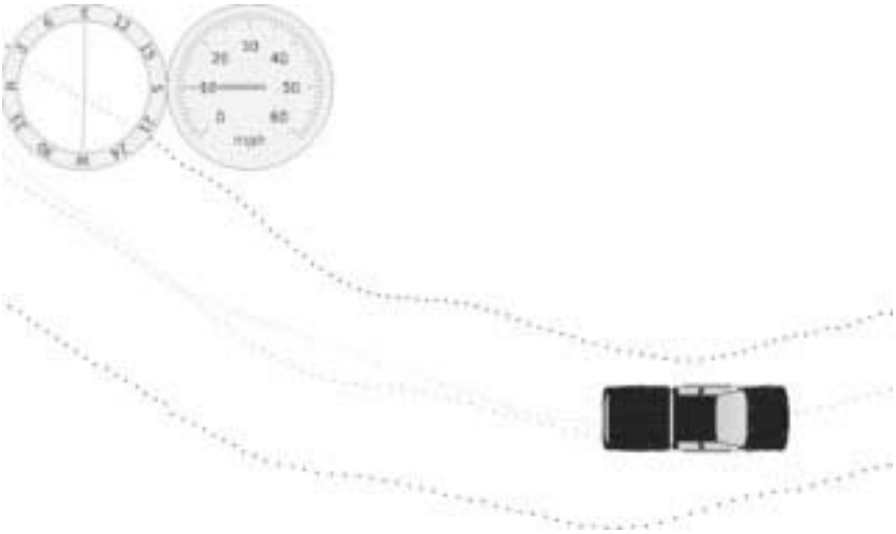

Fig. 7.19. Trail boundaries detected by the Mobileye system

Fig. 7.20. Setup of stereo cameras

7.6.2.1 Hardware Configuration

Figure 7.20 shows the setup of our stereo cameras. Two metal plates sandwich and rigidly fix the cameras so that they can withstand the strong vibrations cased by off-road driving. The distance between the two cameras is 1.2m and each camera is about 1.5m above the ground plane. We use CCD cameras with 7.5mm lenses that have image resolution of 320×240 pixels.

Our stereo system is based on a multi-VLIW processor called *Visconti* [Hattori and Takeda, 2005, Tanabe, 2003]. The processor architecture is designed to ensure efficient performance for general image processing operations while satisfying several requirements for automotive use, e.g., operating temperature range −40 to +85°C, power consumption < 1W@150MHz. Figure 7.21 shows a prototype of an image processing unit using Visconti. It has three video input

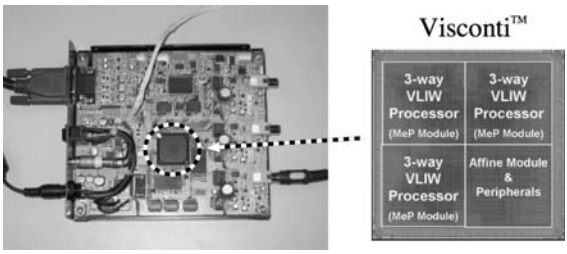

Fig. 7.21. Prototype processing hardware

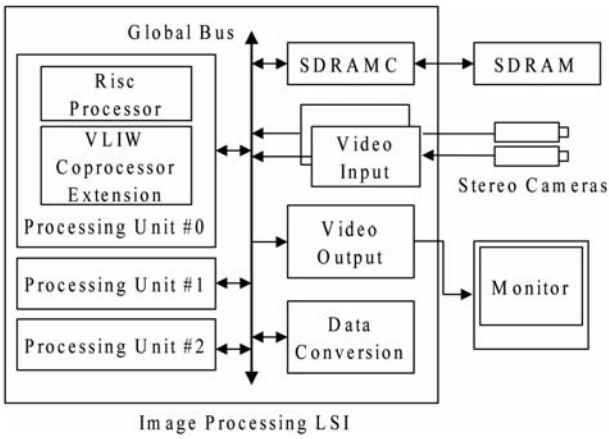

Fig. 7.22. Block diagram of Visconti

channels and a VGA video output to display the processing results. Figure 7.22 shows the block diagram of Visconti. The processor includes one image transformation module and three processing modules operating in parallel. Each of the three processing modules consists of a RISC processor core and a VLIW coprocessor. Several types of SIMD operations required for stereo computation, including convolution, accumulation and pixel shift, are supported in the instruction set of the coprocessor. Each processing module also has a scratch pad memory and a DMA controller so that memory access latency is hidden by double-buffering data translation.

7.6.3 Implementation Details

We adopt sum of absolute differences (SAD) as a matching criterion within a 7×7 window as, SAD is less computationally expensive than other measures such as the sum of squared differences (SSD) and normalized cross correlation (NCC). We also use a recursive technique for efficient estimation of SAD measures

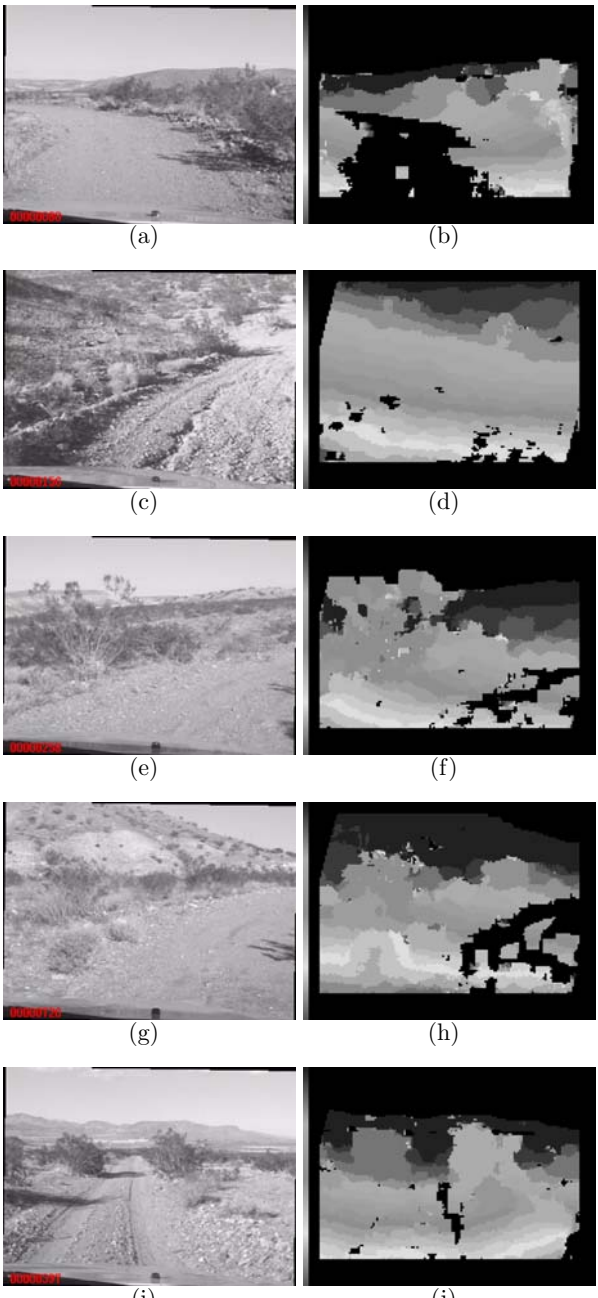

Fig. 7.23. Input images and their disparity maps. More red intensity indicates larger disparity values, i.e., nearer regions, and black indicates texture-less regions. Note that a part of the hood of the vehicle appears on the bottom of input images and these regions are excluded for the disparity estimation.

Faugeras et al. [1993], Hattori and Takeda [2005]. In order to compensate for possible gray-level variations due to different settings of the stereo cameras, the input stereo images are normalized by subtraction of the mean values of the intensities within a matching window at each pixel. Also, the variance of intensities at each point is computed on the reference image to identify those regions which have insufficient intensity variations for establishing reliable correspondences.

As Visconti has one image transformation module and three processing modules operating in parallel, task allocation for these modules is crucial to real-time operation. For instance, the stereo rectification is a indispensable process that transforms input stereo images so that the epipolar lines are aligned with the image scan-lines. The image transformation module carries out the stereo rectification, which is difficult to accelerate by SIMD (single-instruction, multiple-data) operations due to irregular memory access. Also, we divide a pair of images into three horizontal bands which are allocated to those three processing modules. Each of three areas has about the same number of pixels so that the computation cost is equally distributed across the three modules.

Due to the recursive technique for SAD computation, those task allocations and SIMD instructions, our stereo system is capable of disparity estimation at a rate of about 30 frames/sec with up to 30 disparity levels and an image input size of 320×240 pixels (QVGA). This performance is higher than that of a 2.0GHz processor with SIMD instructions. Figure 7.23 shows several examples of disparity images. More red intensity indicates larger disparity values which means closer areas while black indicates texture-less regions. The bottom of input images is excluded from the stereo computation since it corresponds to a part of the hood of the vehicle. These input stereo images were taken in the Mojave desert.

7.7 Results

Golem 2's qualifying runs on the National Qualification Event (NQE) obstacle course were among the best of the field, as shown in Table 7.1, although we also failed on two runs for reasons discussed in Section 7.7.1.

Golem 2 raced out of the start chute at the 2005 Grand Challenge Event (GCE) in the seventh pole position. We knew that Golem 2 was capable of driving well at high speeds. Our speed strategy was that the vehicle would drive at the maximum allowed speed whenever this was below 25 mph. If the recommended speed was greater than 25 mph (implying that the maximum allowed speed was 50 mph) then Golem 2 would exceed the recommended speed, by small amounts at first, but more and more aggressively as the race continued, until it was always driving at the maximum legal speed, except, of course, when modulating its speed during a turn.

As expected, Golem 2 made rapid time on dirt roads and over a dry lakebed. On a paved bridge, Golem 2's laser sensors misperceived the reflective "Botts' dots" in the center of the road as obstacles, which seemed to vanish like a mirage as the vehicle got closer. (See Figure 7.24.) This caused the vehicle to weave back and forth on the bridge, alarming the DARPA observers in the chase vehicle.

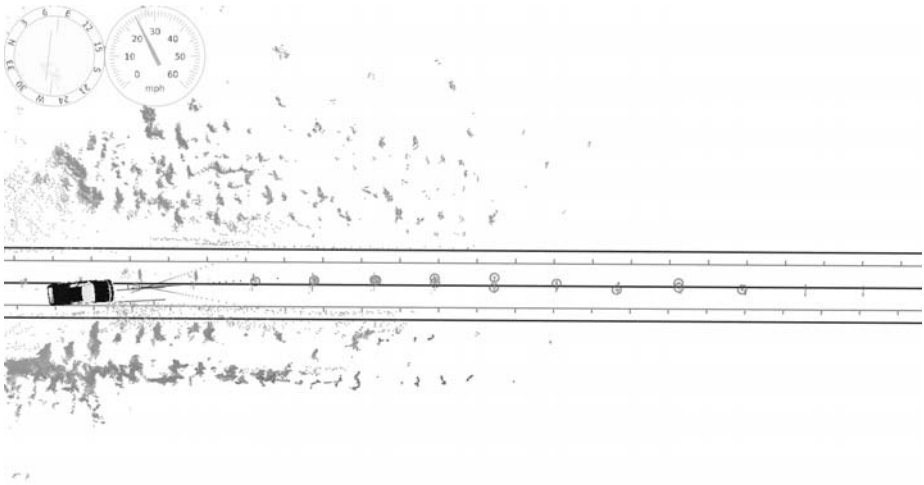

Fig. 7.24. Golem 2 misperceives reflective dots on a paved road surface as obstacles

But our vehicle kept going and once it reached dirt road again, it straightened out and resumed progress at over thirty miles per hour.

The DARPA observers characterized Golem 2 as initially "skittish" and compared it to a teenage driver, but stated that once it left the paved road and entered the desert, they were impressed by its performance and believed they had a winner on their hands.

Unfortunately, after driving twenty-two miles in just under one hour, the computer crashed due to faulty memory management. The uncontrolled vehicle departed from the course boundaries at high speed, crashing through vegetation. The DARPA "pause" button was no longer functional, since no software was running, and the DARPA observers did not press the "disable" button in case the vehicle might recover. Golem 2 hurtled more than half a mile off the course before pounding from the rough terrain finally shook connectors free from its fusebox, killing the engine.

7.7.1 Causes of Failure

Golem 2 crashed on three significant occasions: twice during NQE trials and once during the GCE. We think that all of these failures should be considered mere "bugs" rather than fundamental flaws in the design. Nevertheless it may be interesting to review the causes of these failures.

On its first attempt at the NQE course, Golem 2 immediately veered off to the right and crashed into one side of a gate intended to simulate a cattle crossing. It knocked down the fence beside the gate and came to a stop. The primary cause of this failure was that one of the vertical ladars had been repositioned and miscalibrated (due to a missing decimal point). Mishandling of the course start conditions was a contributing factor.

The sequence of events is illustrated in Figure 7.25. At the 2005 NQE, vehicles were launched out of start chutes which were located far outside the designated course boundaries. We should have been prepared for this special case and the correct procedure was to consider the course boundaries to extend backwards to the start position. However instead Golem 2 reacted as it would generically react to being far off course, by relaxing the course boundary constraints outward. In Figure 7.25(a) the vehicle is moving a straight trajectory which is hardly constrained by the course boundaries. In Figure 7.25(b), the vehicle has moved back onto the RDDF course and also detected the gate ahead where the orange circles indicate obstacles. The planning boundary constraints have contracted inward and the vehicle has planned a trajectory which is very nearly correct, i.e., a trajectory which passes through the first gate. Unfortunately, the boundary constraints have not tightened quite enough and the trajectory skirts the right edge of the RDDF course. In Figure 7.25(c), because of the miscalibrated ladar, the vehicle has misperceived a cloud of phantom obstacles where no obstacles actually exist, and planned to swerve around them to the right. In Figure 7.25(d), the vehicle has perceived that its new trajectory collides with a real obstacle, the fence, but it cannot find a plan that does not appear to result in collision. In Figure 7.25(e), collision is imminent and the vehicle belatedly attempts to brake. In Figure 7.25(f), the fence has been knocked down and the vehicle has come to a stop. Although the vehicle was physically capable of proceding forward over the crushed fence, there was no "restart" logic enabling the vehicle to begin moving again.

Our reaction to this failure was, of course, to fix the calibration of the vertical ladar, correctly handle the special case of starting outside the course boundaries, improve the reaction of the velocity controller to obstacles, and prevent the vehicle from coming to a permanent stop if not paused. We were able to carry out these fixes only because of the very useful `dashboard` visualization tool, shown in Figure 7.25 and elsewhere, that enabled us to closely examine the results of the failed run and simulate what would result from changes to the software.

After two very successful completions of the NQE course, Golem 2 crashed again on the fourth attempt. This time the bug was in the path planner, which failed to properly validate all the possible candidate trajectories and ended up selecting a degenerate trajectory containing two sharp 180-degree turns. The impossibly tight loop in the desired trajectory caused the vehicle to jerk suddenly into a concrete barrier. This event motivated increased evaluation of candidate trajectories, and repair and reinforcement of Golem 2's steering actuator.

Golem 2's final and most distressing failure occurred during the GCE due to static memory overallocation. Large amounts of random access memory were set aside at start time for use in recording obstacles, trajectories, and sensor history. In fact, the memory was overallocated, but this did not become apparent until a large sensor history had accumulated, which, because of the mapping between geographic coordinates and elements of the sensor observation array, only occurred when a large amount of terrain had been covered. Golem 2 had made experimental autonomous runs of 10 miles or so, but had never made a continuous overland journey on the scale of the GCE. Furthermore, an endurance

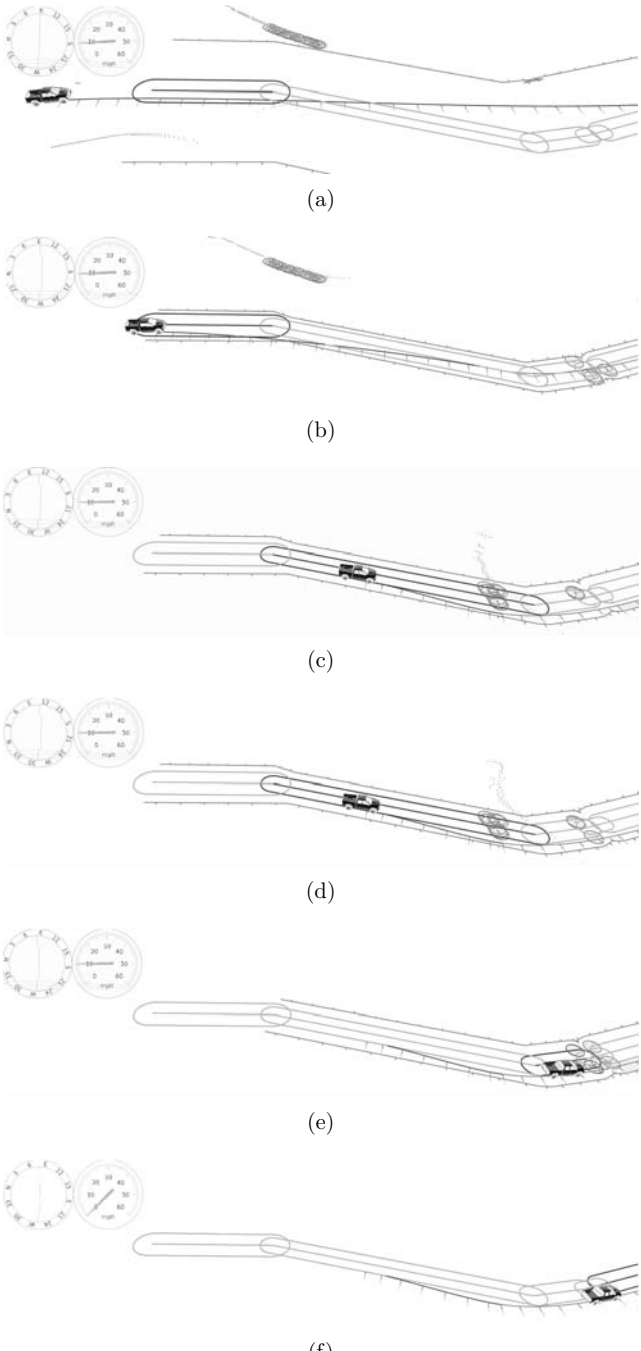

Fig. 7.25. An initial failure of Golem 2

trial which consisted of driving for long periods around a track would probably not have uncovered this bug. Only when Golem 2 had driven across 22 miles of new terrain did the memory bug manifest itself and crash the control program.

Although the Golem 2 software is inherently memory-intensive in its approach, it should be able to operate with well under 1 GB of RAM and therefore this failure was perfectly avoidable in principle.

7.8 Summary and Future Work

Despite occasional problems, the Golem vehicles have demonstrated a high level of high-speed driving performance and we think that our design approach has promise. The system managed to negotiate obstacles at speed using a relatively small amount of computational power (a single 2.2 GHz laptop) and relatively sparse laser range data.

The key drivers of this economically-promising performance include a simplified computational architecture; using a combination of horizontally- and vertically-oriented ladars to reliably sense major obstacles while disregarding inessential details of the terrain; a fast heuristic planner which rapidly finds solutions in typical driving situations; and vehicle state estimation using both an IMU and physical reasoning about the constraints of the vehicle.

The "false" obstacles sometimes announced by the ladar system were not entirely spurious, but generally represented road markings or minor, traversable obstacles. Ideally these would neither be ignored nor mistaken for non-traversable obstacles, but placed in a third category and treated appropriately by the planning software. A future direction for the sensing software is to improve the handling of these features and also to improve the handling of moving obstacles.

Acknowledgments

In addition to the authors of this paper, the Golem Group included Jim Swenson, Jerry K. Fuller, Josh Arensberg, Kerry Connor, Jeff Elings, Izaak Giberson, Maribeth Mason, Brent Morgan, and Bill Caldwell, without whose invaluable technical assistance the Golem vehicles would not have been possible.

The financial support of the Henry Samueli School of Engineering and Applied Sciences at the University of California, Los Angeles, is gratefully acknowledged. In addition, we received financial support and in-kind donations from a number of other organizations, including BEI Technologies, Mobileye, NovAtel, Sick, and OmniStar.

References

Alon, Y., Ferencz, A., and Shashua, A. (2006). Off-road path following using region classification and geometric projection constraints. International Conference on Computer Vision and Pattern Recognition.

Bornstein, J. A. and Shoemaker, C. M. (2003). Army ground robotics research program. In *Unmanned Ground Vehicle Technology V*, pages 303–310.

Coombs, D., Murphy, K., Lacaze, A., and Legowik, S. (2000). Driving autonomously offroad up to 35 km/h. In *Proceedings of the IEEE Intelligent Vehicles Symposium*.

Dickmanns, E. (1997). Vehicles capable of dynamic vision. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1577–1592.

Dickmanns, E. D. (2004). Dynamic vision-based intelligence. *AI Magazine*, 25(2): 10–30.

Faugeras, O., Hots, B., Mathieu, H., Vieville, T., Zhang, Z., Fua, P., Theron, E., Moll, L., Berry, G., Vuillemin, J., Bertin, P., and Proy, C. (1993). Real time correlation-based stereo: algorithm, implementations, and applications. Technical Report Technical report 2013, INRIA, Sophia-Antipolis, France.

Frazzoli, E., Dahleh, M. A., and Feron, E. (2002). Real-time motion planning for agile autonomous vehicles. *AIAA Journal of Guidance, Control, and Dynamics*, 25(1):116–129.

Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156.

Gelb, A. (1974). *Applied Optimal Estimation*. MIT Press, Cambridge, MA.

Hattori, H. and Takeda, N. (2005). Dense stereo matching in restricted disparity space. In *Proceedings of the IEEE Intelligent Vehicles Symposium*, pages 118–123.

Hong, T. H., Abrams, M., Chang, T., and Shneier, M. (2000). An intelligent world model for autonomous off-road driving. *Computer Vision and Image Understanding*.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME*, 82:35–45.

Kalman, R. E. and Bucy, R. S. (1961). New results in linear filtering and prediction theory. *Transactions of the ASME*, 83:95–107.

Kavraki, L., Svestka, P., Latombe, J., and Overmars, M. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In *IEEE Transactions on Robotics and Automation*, volume 12, pages 566–580.

LaValle, S. (1998). Rapidly-exploring random trees: A new tool for path planning. Technical Report TR 98-11, Ames, IA: Iowa State University.

Rasmussen, C. (2002). Combining laser range, color, and texture cues for autonomous road following. In *Proceedings of the IEEE International Conference on Robotics and Automation*.

Rogers, R. M. (2003). *Applied Mathematics in Integrated Navigation Systems*. AIAA Education Series. AIAA, Reston, VA, second edition.

Roundy, D. (2005). *David's Advanced Revision Control System*. `http://darcs.net/`.

Tanabe, J. (2003). Visconti: multi-vliw image recognition processor based on configurable processor. In *Custom Integrated Circuits Conference*, pages 185–188.

Urmson, C. (2005). Navigation regimes for off-road driving. Technical Report CMU-RI-TR-05-23, Carnegie Mellon University Robotics Institute.

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

Welch, G. and Bishop, G. (1995). An introduction to the kalman filter. Technical Report TR 95-041, University of North Carolina at Chapel Hill, Chapel Hill, NC.