Mapping with an Autonomous Car

Pierre Lamon* Cyrill Stachniss* Rudolph Triebel† Patrick Pfaff†
Christian Plagemann† Giorgio Grisetti† Sascha Kolski* Wolfram Burgard† Roland Siegwart*

**Eidgenössische Technische Hochschule (ETH), Inst. of Robotics and Intelligent Systems, 8092 Zurich, Switzerland*
†University of Freiburg, Department of Computer Science, D-79110 Freiburg, Germany

***Abstract*— In this paper, we present an approach towards mapping and safe navigation in real, large-scale environments with an autonomous car. The goal is to enable the car to autonomously navigate on roads while avoiding obstacles and while simultaneously learning an accurate three-dimensional model of the environment. To achieve these goals, we apply probabilistic state estimation techniques, network-based pose optimization, and a sensor-based traversability analysis approach. In order to achieve fast map learning, our system compresses the sensor data using multi-level surface maps. The overall system runs on a modified Smart car equipped with different types of sensors. We present several results obtained from extensive experiments which illustrate the capabilities of our vehicle.**

I. I

Learning models of the environment and safely navigating based on that models is a fundamental task of mobile robots. Many researcher focused on learning map of indoor as well as outdoor scenes. Recently, modified cars became a new platform in robotics. Compared to standard robots, cars offer the possibility to travel longer distances, carry more sensors, and thus being more suitable for mapping large areas. On the one hand, this offers the opportunity to address robotic tasks on a larger scale but on the other hand requires efficient solutions to common problems like mapping or localization.

In this paper, we describe our modified Smart car equipped with different sensors to monitor the environment. We present our approach to mapping large outdoor areas with that vehicle. Our map representation can be seen as an extension of elevation maps that allows us to model different layers in the environment and therefore different drivable areas like, for example, bridges or underpasses. It overcomes serious limitations of elevation maps and is able to model the environment in an adequate way needing only a few more memory resources compared to elevation maps.

The reminder of this paper is organized as follows. After the discussion of related work, we present detail about our modified Smart car. Then, we will explain our approach to localization using different sensors. In Section V, we introduce our model of the environment and present a method to learn this model. Finally, we present experiments illustrating the maps obtained with our robot in real world experiments.

II. R W

The problem of learning models of the environment has been studied intensively in the past. Most approaches generate two-dimensional models from range sensor data and a series of different approaches have been developed [22, 24, 37, 8, 7, 40, 11, 19]. In the literature, those approaches are often referred to as solutions to the simultaneous localization and mapping (SLAM) problem. Recently, several techniques for acquiring three-dimensional data with 2d range scanners installed on a mobile robot have been developed. A popular approach is to use multiple scanners that point towards different directions [41, 13, 42]. An alternative is to use pan/tilt devices that sweep the range scanner in an oscillating way [32, 26]. More recently, techniques for rotating 2d range scanners have been developed [17, 50].

Many authors have studied the acquisition of three-dimensional maps from vehicles that are assumed to operate on a flat surface. For example, Thrun *et al.* [41] present an approach that employs two 2d range scanners for constructing volumetric maps. Whereas the first is oriented horizontally and is used for localization, the second points towards the ceiling and is applied for acquiring 3d point clouds. Früh and Zakhor [10] apply a similar idea to the problem of learning large-scale models of outdoor environments. Their approach combines laser, vision, and aerial images. Furthermore, several authors have considered the problem of simultaneous localization and mapping (SLAM) in an outdoor environment [5, 12, 43]. These techniques extract landmarks from range data and calculate the map as well as the pose of the vehicles based on these landmarks. Our approach described in this paper does not rely on the assumption that the surface is flat. It uses surface maps to capture the three-dimensional structure of the environment and is able to estimate the pose of the robot in all six degrees of freedom.

One of the most popular representations are raw data points or triangle meshes [1, 20, 32, 44]. Whereas these models are highly accurate and can easily be textured, their disadvantage lies in the huge memory requirement, which grows linearly in the number of scans taken. Accordingly, several authors have studied techniques for simplifying point clouds by piecewise linear approximations. For example, Hähnel *et al.* [13] use a region growing technique to identify planes. Liu *et al.* [21] as well as Martin and Thrun [23] apply the EM algorithm to cluster range scans into planes. Recently, Triebel *et al.* [46] proposed a hierarchical version that takes into account the parallelism of the planes during the clustering procedure. An alternative is to use three-dimensional grids [27] or tree-based representations [34], which only grow linearly in the size of the environment. Still, the memory requirements for such maps in outdoor environments are high.

In order to avoid the complexity of full three-dimensional maps, several researchers have considered elevation maps as an attractive alternative. The key idea underlying elevation maps is to store the $2\frac{1}{2}$-dimensional height information of the terrain in a two-dimensional grid. Bares *et al.* [2] as well as Hebert *et al.* [14] use elevation maps to represent the environment of a legged robot. They extract points with high surface curvatures and match these features to align maps constructed from consecutive range scans. Singh and Kelly [36] extract elevation maps from laser range data and use these maps for navigating an all-terrain vehicle. Ye and Borenstein [51] propose an algorithm to acquire elevation maps with a moving vehicle carrying a tilted laser range scanner. They propose special filtering algorithms to eliminate measurement errors or noise resulting from the scanner and the motions of the vehicle. Wellington *et al.* [49] construct a representation based on Markov Random Fields. They propose an environment classification for agricultural applications. They compute the elevation of the cell depending on the classification of the cell and its neighbors. Compared to these techniques, the contribution of the mapping approach presented in this paper lies in two aspects. First, we classify the points in the elevation map into horizontal points seen from above, vertical points, and gaps. This classification is important especially when a rover is deployed in an urban environment. In such environments, typical structures like the walls of buildings cannot be represented in standard elevation maps. Second, we describe how this classification can be used to improve the matching of different elevation maps.

In the context of autonomous cars, a series of successful systems [45, 3, 48] have been developed for the DARPA Gran Challenge [4], which was a desert race for autonomous vehicles along an approximatively 130 mile course. As a result of this challenge, there exist autonomous cars that reliably avoid obstacle and navigate at comparably high speeds. The focus of the Gran Challenge was to finish the race as quickly as possible whereas certain issues like building consistent large-scale maps of the environment have been neglected since they where not needed for the race. Our approach towards mapping large areas therefore has a different aim compared to the vehicles participating to the Gran Challenge. Nevertheless, our Smart car also benefited from different techniques used within the Gran Challenge. We apply a similar approach to follow a given trajectory than the winning vehicle Stanley [45].

III. V D

Our vehicle, called SmartTer (Smart all Terrain), is a standard Smart car that has been enhanced for fully autonomous driving in both urban and non-urban environments. The model is a Smart fortwo coupé passion of year 2005, which is equipped with a 45 kW engine. This model has been chosen because it gathers several advantages:

- *compact and light weight* Such characteristics allow us to easily transport the vehicle on a trailer to the testing area[1] and fits in our lab's mechanical workshop. Furthermore, its light weight yields fair locomotion performance in rough terrain.
- *power steering* The power steering motor can deliver enough torque to steer the car. So, it is possible to "steer by wire" with minor modification.
- *auto gearshift* No additional modification is required to switch gears while the car is driving.
- *easy access to the CAN bus* Important sensory information such as steering wheel angle and wheels velocities and car status are directly accessible.

All these features facilitate the process of modifying such a vehicle for autonomous driving. The fully equipped SmartTer is depicted in Fig. 1 and 2.

[1]Because the car has been deeply modified, it is not allowed to drive on public streets.

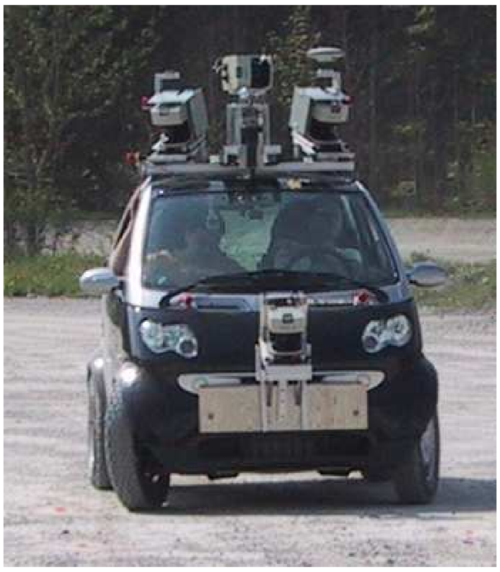
Fig. 1. SmartTer front view

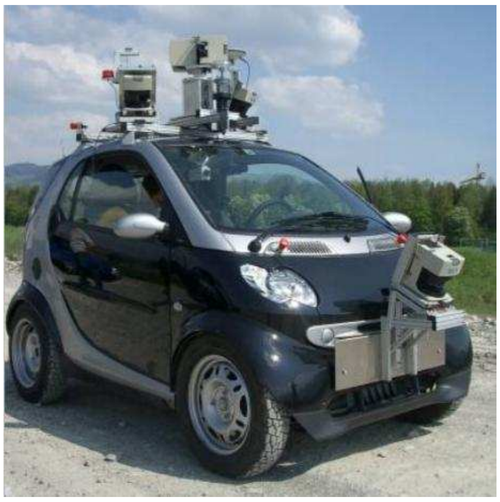
Fig. 2. SmartTer side view

A. Vehicle modifications

In order to enhance the original model for autonomous driving, several modifications have been performed on the car. This section describes the mechanical and electrical changes that were necessary.

• Wheels with better grip and larger diameter have been mounted. This yields to a higher ground clearance and much better traction in rough terrain.

• A 24V power generator has been installed in order to power all the electronic devices and additional actuators. The generator is driven by a belt and pulley that is directly connected to the engine output axis (which is situated under the trunk, at the rear of the car). Two batteries placed in the trunk act as an energy buffer. They have a total capacity of 48Ah and are continuously recharged when the engine is running.

• To provide a clean interface to the vehicle, we integrated an automotive ECU designed for highly reliable real-time applications. This ECU has four CAN interfaces as well as a wide variety of analog and digital I/O. In the vehicle, it is sitting between the computers on the one hand and the vehicle CAN bus and our actuators on the other hand. In this ECU, we implemented a state machine that allows us to enable different modes of operation (STOP, PAUSE and RUN) via wired buttons as well as by a wireless remote control. Besides these emergency buttons, the ECU handles timeouts in the command coming from the computers and ensures a safe vehicle state whenever those commands are missing.

• The power steering system applies a torque M_{add} on the steering column that allows to minimize the effort required by the driver to steer the wheels (see Fig. 3). This task is fulfilled by the driving assistance unit (DA unit), which minimize the torque sensed in the steering column by applying an appropriate voltage to the power steering motor. The gain of the DA controller is set based on the car's velocity, which is broadcasted on the CAN bus of the vehicle. In order to use the power steering motor for "steering by wire", a specific electronic board has been designed and inserted in the vehicle's control loop. The Vehicle CAN bus was disconnected from the DA and routed to a computer (Rack0 in Fig. 3) so that the steering angle a_s can be read and used by our own controller. An additional CAN bus, called Computer CAN, allows to feed the DA unit with the minimal set of CAN messages required for the proper operation of the unit i.e. engine status and car velocity messages. The same bus is used to send commands to a CAN to analog module which "fakes" the torque voltage needed by the DA unit. Finally, the steering angle is controlled with a PID controller running on the control computer, which minimizes the steering angle error $e = a_t - a_s$, where a_t is the desired steering angle. A switch mounted next to the steering wheel, enables the selection of manual or controlled mode.

• A system of cable and pulleys is used to activate the break pedal. The servo motor that pulls the cable is placed under the driver's seat and is commanded using the Computer CAN.

• A dedicated electronic board has been developed to enable the use of a computer to set the gas command. The command voltage, originally provided by a potentiometer embedded in the gas pedal, is simply generated by a CAN to analog device. This device receives commands through the Computer CAN bus. A button mounted inside the car allows us to choose between normal or controlled mode.

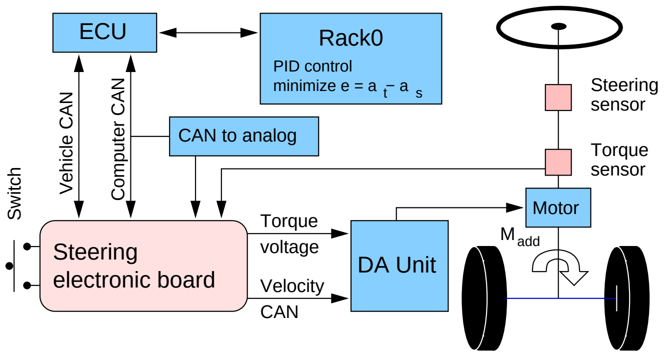

Fig. 3. Steer by wire system

B. Sensors

The sensors used for outdoor applications must meet strong requirements such as mechanical robustness, water/dust proofness and limited sensitivity to sun light. Thus, in comparison with indoor applications, the choice is limited and the selection of optimal sensors must be done carefully. In this section, all the sensors that have been mounted on the car are described.

• *Three navigation laser scanner sensors* (SICK LMS291-S05, outdoor version, rain proof, low sensitivity to sun light) These sensors are used mainly for obstacle avoidance and local navigation. One sensor is placed at the lower front slightly looking down and the two others on the roof, looking to the sides and slightly down. This 'A' shape configuration enables a large field of view and is well adapted to all kind of terrains. The three sensors are visible in Fig. 1 and Fig. 4 depicts a closer view of the sensors mounted on the roof.

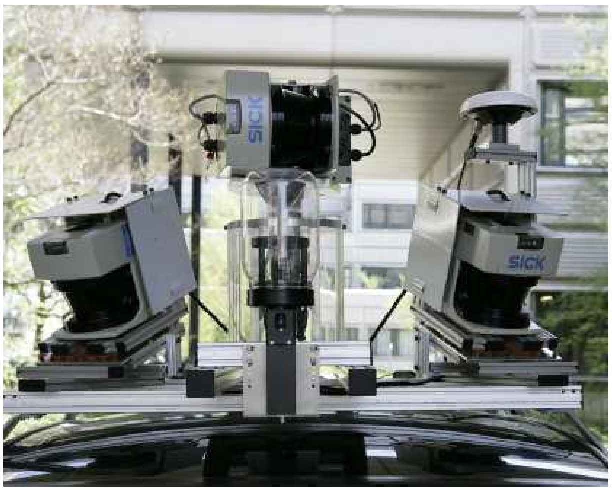

Fig. 4. Sensors mounted on the roof

• *Rotating 3D scanner* This is a custom made sensor that is mounted on the roof (see Fig. 5). In order to acquire 3D scans of the environment, two SICK LMS291-S05 are mounted sideways on a plate rotating around the vertical axis. A signal is triggered each time the plate performs a full turn. That way it is possible to know its angular position at each time (using the rotational velocity of the motor and the elapsed time since the last trigger). The data and power lines of the two SICK sensors go through a slip ring, which is mounted along the rotation axis. The 3D scans are mainly used to compute a consistent 3D digital terrain model of the environment.

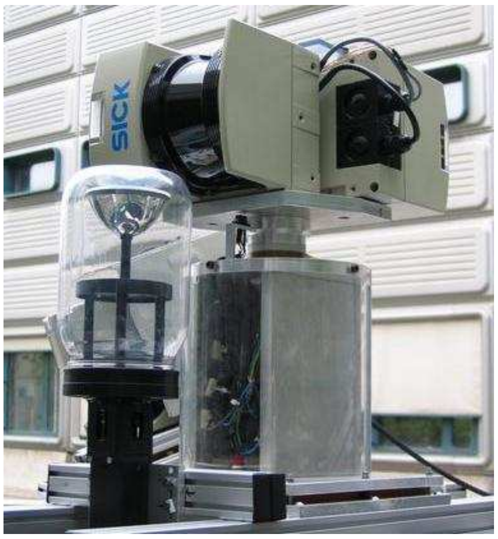
Fig. 5. Rotating scanner and omnidirectional camera

• *Omnidirectional camera* (Sony XCD-SX910CR, focal length 12mm, with parabolic mirror and dust protection) The setup is mounted in front of the rotating scanner (see Fig. 5). It enables the acquisition of panoramic images that are used to supply texture information for the 3D digital terrain maps. The images are also exploited to detect artificial object in the scene.
• *Monocular camera* (Sony XCD-SX910CR, focal length 4.2mm) This camera is placed in the car, behind the wind shield. Like the omnidirectional camera, it is used to detect artificial object in the scene.
• *Differential GPS system* (Omnistar Furgo 8300HP) This device provides the latitude, longitude and altitude together with the corresponding standard deviation and the standard NMEA messages with a frequency of 5 Hz. When geostationary satellites providing the GPS drift correction are visible from the car, the unit enters the differential GPS mode (high precision GPS). When no correction signal is available, the device outputs standard precision GPS.
• *Car sensors* The measurements taken by the car sensors are reported with a frequency of 100 Hz and are accessible via the CAN bus of the vehicle. The car provides motor RPM, temperatures, steering wheel angle, wheel velocities, gas pedal position, and some further status information.
• *Optical gyroscope* (KVH DSP3000) This fibre optic gyroscope can measure very low rotation rates with a frequency of 100 Hz. It is possible to use it as a heading sensor for a comparably long period of time by integrating the angular rate. Contrary to compasses, the integrated heading is not sensitive to earth magnetic field disturbances. Finally, this unit offers much better accuracy than mechanical gyro and is not sensitive to shocks because it contains no moving parts.
• *Inertial measurement unit* (Crossbow NAV420, waterproof) This unit provides sensor data with a frequency of 100 Hz that contains the measurements from 3 accelerometers, 3 gyroscopes, a 3D magnetic field sensor and a GPS receiver. The internal digital signal processor of the unit combines the embedded sensors to provide the filtered attitude of the vehicle (roll, pitch, heading to true north) and the position (latitude, longitude and altitude). However, this sensor is not well adapted for a ground vehicle driving at low speed because of a bad signal/noise ratio of the inertial sensors. Furthermore, the earth magnetic field can be strongly distorted when the vehicle drives next to iron structures. This causes large error on the estimated heading. For better accuracy and robustness, we implemented our own localization algorithm, which is presented in section IV. The algorithm combines the measurements taken by the IMU, the differential gps, the car sensors and the optical gyroscope.

C. *Computational power and software architecture*

The system consists of four compact PCI computer racks communicating trough a gigabit Ethernet link. All the racks have the same core architecture which is a Pentium M processor running at 2GHz, equipped with 1.5GB of RAM, Gigabit and Fast Ethernet, two RS232 serial ports, USB ports and a 30GB hard disk. Each rack is dedicated to specific tasks and acquires measurement from different sensors as it is depicted in Fig. 6.

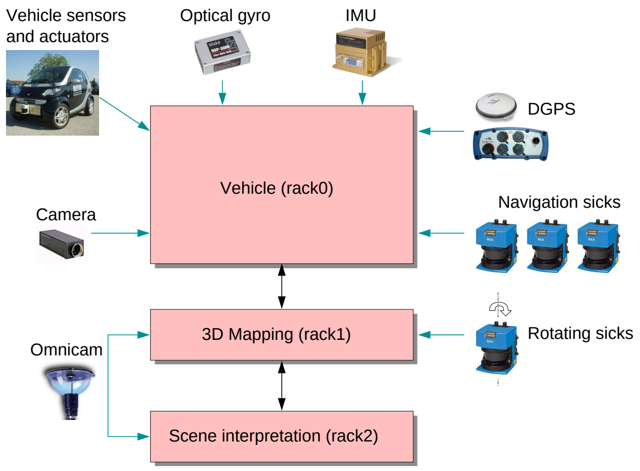

Fig. 6. Architecture of the system

The computer racks run the Linux operating system and the software architecture is based on both GenoM [9] and Carmen [25] robotic software architectures. The functional modules running on different computers exchange data using the Inter Process Communication (IPC) [35]. To guarantee that the time stamp associated to each data packet is globally consistent, the CPU clocks are synchronized using the Network Time Protocol Daemon (NTPD). In order to reduce communication delays, the architecture has been designed in such a way that it minimizes the amount of transmitted data.

The *Vehicle rack* is endowed with a CAN interface that is used to access the vehicle CAN bus. The measurements of the internal car sensors are continuously read and the car commands such as the vehicle velocity and steering angle are passed to the ECU. The other sensors used for localization i.e. the DGPS, the IMU and the optical gyro are connected to the rack through RS232 serial ports. Finally, the measurements of the three navigation scanners are acquired through high speed RS422 serial ports and the images of the navigation camera grabbed using an IEEE1394 interface. The main tasks of the Vehicle rack are a) to keep track of the vehicle position b) to control its motion (steering, breaking, velocity control) and c) obstacle avoidance and path planning.

A 3D map of the traversed environment is updated on the *3D Mapping rack* using the measurements acquired by the rotating 3D scanner. Like on the Vehicle rack, the scanner data is acquired through RS422 ports. The index signal is detected using a multi-purpose IO board and the motor speed is set using an RS232 interface. For more realistic rendering, the texture information acquired by the omnidirectional camera is mapped on the 3D model as it is depicted in Fig. 7

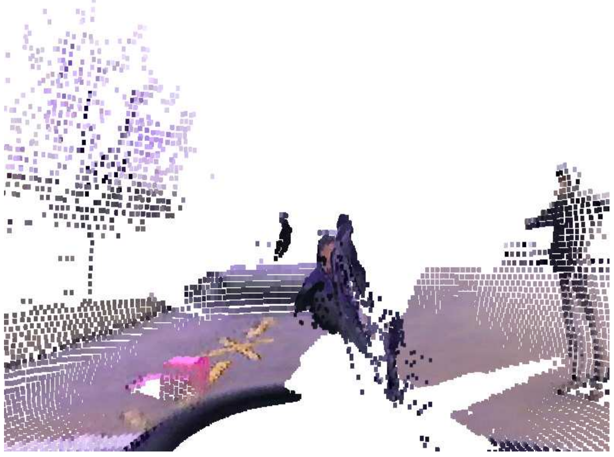

Fig. 7. 3D scan with texture. One recognize a tree on the left hand side, yellow street markings (-x- shape), a pink box (next to the street markings) and persons (e.g. on the right, with arms extended)

Finally, the scene analysis is performed on the *Scene interpretation rack*. The artificial objects are extracted from the textured 3D maps and raw omnicam images and their representation and location are stored in memory as the vehicle moves along the path.

IV. L

Our localization algorithm is based on the inverse form of the Kalman filter, i.e., the information filter. This filter has the property of summing information contributions from different sources in the update stage. This characteristic is advantageous when many sensors are involved, which is the case in our system. The localization is done in two steps, namely the state prediction and the state update.

A. State Update

The car state is updated using the measurements taken by several complementary sensors.

- *Differential GPS* We use the WGS-84 standard to convert the GPS coordinates in Cartesian coordinates (x, y, z) expressed in a local navigation frame n. The heading to true north ψ is also output by the unit and is available in the RMC message. The measurement model for the GPS is

$$z_{gps} = \begin{bmatrix} x_{gps} \\ y_{gps} \\ z_{gps} \\ \psi_{gps} \end{bmatrix}_n = \begin{bmatrix} x \\ y \\ z \\ \psi \end{bmatrix}_n + v_{gps} \tag{1}$$

In order to reject the erroneous fixes caused by multi-path and satellite constellation changes, we use the following gating function [39]

$$z^T(k) \cdot S^{-1} \cdot z(k) \leq \gamma, \tag{2}$$

where S is the innovation covariance of the observation. The value of γ is set to reject innovations exceeding the 95% threshold.

- *Car sensors* For localization, we use the velocity $\dot{x}_{odo}$ of the car from the CAN bus. Unlike in the case of a flight vehicle, the motion of a wheeled vehicle on the ground is governed by nonholonomic constraints. Under ideal conditions, there is no side slip and no motion normal to the ground surface: the constraints are written as $\dot{y}_{odo} = 0$ and $\dot{z}_{odo} = 0$. In any practical situation, these constraints are often violated. Thus, as in [6], we use zero mean Gaussian noise to model the extent of constraint violation. The measurement model for the odometry is then expressed as

$$z_{odo} = \begin{bmatrix} \dot{x}_{odo} \\ 0 \\ 0 \end{bmatrix}_b = \left[C_b^n\right]^T \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}_n + v_{odo}, \tag{3}$$

where C_b^n is the matrix for transforming velocities expressed in the car's frame b into the navigation frame n. The observation noise covariance is obtained using

$$R_{odo} = C_b^n \cdot diag\left\{\sigma^2_{enc}, \sigma^2_{vy}, \sigma^2_{vz}\right\}\left[C_b^n\right]^T, \tag{4}$$

where σ^2_{enc} is the variance of the car velocity and σ^2_{vy},σ^2_{vz} are the amplitude of the noise related to the constraints.

- *Optical gyroscope* The measurement model for the optical gyroscope is

$$z_{opt} = \psi_{opt} = \psi + b_{opt} + v_{opt}, \tag{5}$$

where b_{opt} is the angular offset between the heading to true north ψ and the actual measurement of the gyro.

- *Inertial measurement unit* For the reasons mentioned before, we disabled the GPS and used the unit in angle mode: the unit outputs the filtered roll, pitch and heading to magnetic north. The measurement model for this sensor is

$$z_{imu} = \begin{bmatrix} \phi_{imu} \\ \theta_{imu} \end{bmatrix}_n = \begin{bmatrix} \phi \\ \theta \end{bmatrix}_n + v_{imu} \tag{6}$$

$$\psi_{imu} = \psi + b_{imu} + v_{himu}, \tag{7}$$

where b_{imu} is the angular offset between ψ and the heading measurement provided by the IMU.

B. Prediction model

We apply a standard prediction model for the car which has the following form

$$\mathbf{x}_{k+1} = \begin{bmatrix} F_x & \dots & & 0 \\ \vdots & F_y & & \\ & & F_z & \vdots \\ 0 & & \dots & I_{5x5} \end{bmatrix} \cdot \mathbf{x}_k + w_k. \tag{8}$$

The state vector $\mathbf{x}$ contains the position and velocity expressed in the navigation frame n, the orientation of the vehicle represented by the three angles roll ϕ, pitch θ and yaw ψ and the two biases b_{imu} and b_{opt} :

$$\mathbf{x} = \begin{bmatrix} x & \dot{x} & y & \dot{y} & z & \dot{z} & \phi & \theta & \psi & b_{imu} & b_{opt} \end{bmatrix}^T \tag{9}$$

The position of the vehicle at time $k+1$ is predicted using the position and velocity at time k. This takes the form of a first order process written as

$$F_{x,y,z} = \begin{bmatrix} 1 & h \\ 0 & 1 \end{bmatrix}_k \tag{10}$$

where h denotes the sampling period ($h = 10\,\text{ms}$). All the other elements of the state vector are predicted as simple Gaussian processes. The covariance matrix Q_k associated to the state prediction process is represented as

$$Q_k = G_k \cdot q_k \cdot G_k^T \tag{11}$$

where q_k is a diagonal matrix containing the variances of the elements of the state vector

$$q_x = diag\left\{ \sigma_x^2 \quad \sigma_y^2 \quad \sigma_z^2 \quad \sigma_\phi^2 \quad \sigma_\theta^2 \quad \sigma_\psi^2 \quad \sigma_{b_{imu}}^2 \sigma_{b_{opt}}^2 \right\} \tag{12}$$

Finally, the matrix mapping the noise covariance q_k to the process covariance Q_k is written as

$$G_k = \begin{bmatrix} g_x & \dots & & 0 \\ \vdots & g_y & & \vdots \\ & & g_z & \\ 0 & & \dots & diag_{5x5}(h) \end{bmatrix}_k \tag{13}$$

where

$$g_{x,y,z} = \begin{bmatrix} h^2/2 \\ h \end{bmatrix} \tag{14}$$

V. M B

A. Map Representation

To represent the 3D data acquired with the rotating laser scanners, we use Multi-Level Surface (MLS) maps [47]. These maps can be regarded as an extension to elevation maps [2, 14, 36, 31, 29, 18]. The idea here is that each cell in a 2D grid can contain many representations of 3D objects called *surface patches*. A surface patch consists of a mean μ and a variance σ, as well as a *depth value d*. Here, μ and σ define a Gaussian distribution that reflects the uncertainty of the measured height of an object's surface. The depth value d represents the length

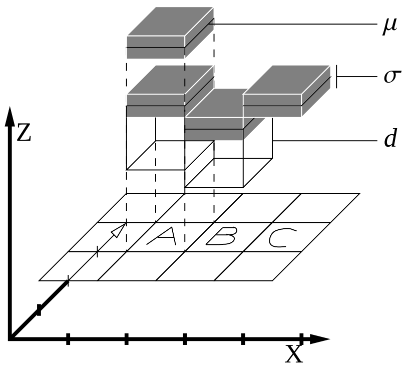

Fig. 8. Example of different cells in an MLS Map. Cells can have many surface patches (cell A), represented by the mean and the variance of the measured height. Each surface patch can have a depth, like the patch in cell B. Flat objects are represented by patches with depth 0, as shown by the patch in cell C.

of a vertical interval starting at μ and pointing downwards. The motivation behind this is to represent flat objects, such as street, ground etc., and non-flat objects such as buildings in the same framework. A surface patch of a building will usually have a large depth value, while the depth of a street patch is in general 0. Figure 8 illustrates different possible surface patches in a MLS map.

B. Traversability Analysis and Feature Extraction

One main goal of the MLS map representation is the ability to classify the terrain in which the robot moves. This classification is important to use the map for path planning. Another design goal for the MLS maps was the possibility to match local MLS maps to one big map, without relying on the raw point cloud data. This matching process is usually performed using the iterative closest point algorithm (ICP). The map matching using ICP has been shown to be very efficient when applying it to subsets of features rather than to the entire data set [28, 33]. Therefore, we first classify the surface patches into the three classes 'traversable', 'non-traversable' and 'vertical'. Then we subsample each of these classes and look for corresponding patches in the other map. This will be described in the next section. For the patch classification, we define vertical patches as those having non-zero depth values. A patch is considered as traversable if it is flat (the depth is 0) and the distance between its height and the height of the neighboring patches does not exceed 10*cm*. All other flat patches are classified as non-traversable.

C. Map Matching

To calculate the alignments between two local MLS maps calculated from individual scans, we apply the ICP algorithm. The goal of this process is to find a rotation matrix R and a translation vector $\mathbf{t}$ that minimize an appropriate error function. Assuming that the two maps are represented by a set of Gaussians, the algorithm first computes two sets of feature points, $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_{N_1}\}$ and $\mathcal{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_{N_2}\}$. In a second step, the algorithm computes a set of C index pairs or *correspondences* $(i_1, j_1), \dots, (i_C, j_C)$ such that the point $\mathbf{x}_{i_c}$

in $\mathcal{X}$ corresponds to the point $\mathbf{y}_{j_c}$ in $\mathcal{Y}$ for $c = 1, \ldots, C$. Then, in a third step, the error function e defined by

$$e(R, \mathbf{t}) := \frac{1}{C} \sum_{c=1}^{C} (\mathbf{x}_{i_c} - (R\mathbf{y}_{j_c} + \mathbf{t}))^T \Sigma^{-1} (\mathbf{x}_{i_c} - (R\mathbf{y}_{j_c} + \mathbf{t})), \quad (15)$$

is minimized. Here, Σ denotes the covariance matrix of the Gaussian corresponding to each pair $(\mathbf{x}_i, \mathbf{y}_i)$. In other words, the error function e is defined by the sum of squared Mahalanobis distances between the points $\mathbf{x}_{i_c}$ and the transformed point $\mathbf{y}_{j_c}$. In the following, we denote this Mahalanobis distance as $d(\mathbf{x}_{i_c}, \mathbf{y}_{j_c})$.

In principle, one could define this function to directly operate on the Gaussians when aligning two different MLS maps. However, this would result in a high computational effort, especially if the maps are very big and many Gaussians are stored. Additionally, we need to take care of the problem that the intervals corresponding to vertical structures vary substantially depending on the view-point. Moreover, the same vertical structure may lead to varying heights in the surface map when sensed from different locations. In practical experiments, we observed that this introduces serious errors and often prevents the ICP algorithm from convergence. To overcome this problem, we separate Eq. (15) into three components each minimizing the error over the individual classes of points. These three terms correspond to the three individual classes, namely surface patches corresponding to vertical objects, traversable surface patches, and non-traversable surface patches.

Let us assume that $\mathbf{u}_{i_c}$ and $\mathbf{u}'_{j_c}$ are corresponding points, extracted from vertical objects. The number of points sampled from every interval classified as vertical depends on the height of this structure. In our current implementation, we typically uniformly sample four points per meter. The corresponding points $\mathbf{v}_{i_c}$ and $\mathbf{v}'_{j_c}$ are extracted from traversable surface patches, $\mathbf{w}_{i_c}$ and $\mathbf{w}'_{j_c}$ are extracted from not traversable surfaces. The resulting error function then is

$$e(R, \mathbf{t}) = \underbrace{\sum_{c=1}^{C_1} d_v(\mathbf{u}_{i_c}, \mathbf{u}'_{j_c})}_{\text{vertical cells}} + \underbrace{\sum_{c=1}^{C_2} d(\mathbf{v}_{i_c}, \mathbf{v}'_{j_c})}_{\text{traversable}} + \underbrace{\sum_{c=1}^{C_3} d(\mathbf{w}_{i_c}, \mathbf{w}'_{j_c})}_{\text{non-traversable}}. \quad (16)$$

In this equation, the distance function d_v calculates the Mahalanobis distance between the lowest points in the particular cells. To increase the efficiency of the matching process, we only consider a subset of these features by sub-sampling.

D. Loop Closing

Usually, the map matching process described in the previous section works well in cases where the robot travels only a short distance. However, for longer distance the accumulated local matching errors may be so large, that the overall map becomes inconsistent. This becomes visible especially when the robot returns to areas where it has been before, which is usually called *loop closing*. In our case, this matching error is bounded due to the high accurate GPS based global localization described before. However, a smaller matching error still remains and is visible in the maps. Therefore, we apply a global pose estimation technique similar to the one presented in [30]. For the details of this technique we refer to [47]. We only note that it is based on a non-linear minimization of the difference between the 3D transformation parameters $(x, y, z, \varphi, \vartheta, \psi)$ resulting from the robot poses and those given by local pose constraints between overlapping local maps. The local pose constraints are obtained by applying ICP matching between overlapping local maps.

VI. P P

In order to acquire data about the environment, the car needs to drive through its environment and visit the different locations. This can be done by manually driving the car or in a more challenging way by autonomous navigation. A realistic approach is to provide a route description to the car and let it run autonomously along that route. However, it is not sufficient for safe navigation to only follow a predefined route since obstacle might block the route and the car has to plan an admissible trajectory around them.

The current version of our planning system follows the ideas of Kelly and Stentz [15] and is closely related to the approach of Thrun *et al.* [45]. The idea is to generate variations of the originally specified route. The robot then evaluates the different trajectories and selects the best admissible one given a cost function. The trajectories are evaluated according to traversability, curvature, and alignment with the specified route. The chosen trajectory is then sent to a low level controller, that keeps the car on the selected trajectory. The controller itself does not change the speed of the vehicle, it only adapts the steering angle of the car. The bigger the error between the car and the trajectory it should follow, the more the car tries to steer towards the given trajectory. We started with a controller that was also applied by Thrun et al. [45] which is given by the control low

$$\alpha(t) \quad = \quad \phi(t) + atan\left(\kappa \cdot \frac{x(t)}{v(t)}\right), \quad (17)$$

where α refers to the new steering command, $\phi(t)$ to the difference of the current steering angle and the orientation of the trajectory the car should follow. $x(t)$ refers to the current distance between the position of the robot and the trajectory and $v(t)$ describes the velocity of the car. The control gain κ influences, how intensive the car steers back to the trajectory. A too high value leads to oscillation and a too small value leads to a comparably slow convergence rate to the reference trajectory. We determined the gain through experiments and obtain good results for $\kappa \in [0.2, 0.4]$.

This controller follows the given trajectory but can lead to small overshoots in curves and to slightly shaky steering commands of the car. To overcome this problem, we do not use the raw velocity information but apply a post filtering in a Kalman filter fashion. This leads to smoother velocity estimates and helps to stabilize the steering command. We furthermore do not compute $\phi(t)$ only based on the closest point on the route but rather averaging of a few poses in front of the car. This compensates for slightly un-smooth input trajectories and reduces the risk of slight overshoots in curves.

VII. S T

The need for a sophisticated simulation environment for our autonomous vehicle mainly originated in two facts. First, the complex software architecture developed by several researchers at different labs had to be tested as a complete system as early as possible to verify the appropriateness of interfaces, interaction protocols, and data rates. Second, as the development of software and hardware was conducted in parallel from the early beginning of the project on, there was an inherent need for physically plausible data sets to test the algorithms, especially in the area of 3d mapping and navigation.

As a consequence, we have built a 3d simulation environment for our autonomous vehicle, its main sensors, and the outdoor terrain. An example for such a simulation is depicted in Figure 9. The developed system is based on the Gazebo simulator [16], which is part of the Player/Stage project. Gazebo uses the Open Dynamics Engine [38] to yield physically plausible simulations in three dimensions by taking into account friction, forces, and rigid body dynamics. It includes a wide variety of pre-build models for robotics applications and is relatively easy to extend. We developed a number of plug-in models for our autonomous smart car and its primary sensors which are three static laser range finders, two rotating laser range finders, a GPS and inertial sensor.

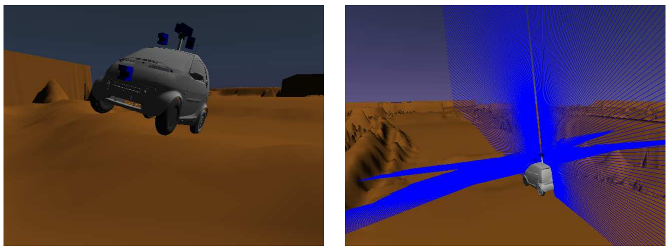
Fig. 9. The simulated autonomous Smart car in outdoor terrain (left). The simulation includes two rotating laser range finders mounted on top of the vehicle (right). The laser beams are visualized in blue.

Instances of these models can be configured and inserted into a simulated world using a simple XML-based description language. At the same time, the plug-in models implement the necessary interfaces and IPC-based communication protocols to our navigation and mapping system so they can readily be exchanged for real hardware components. There are no simulator specific message types build into the overall system to ensure that the architecture is clearly focused on the real application domain.

A. The Simulated Environment

In general, there are several different ways how the simulated environment can be specified. For testing basic navigation capabilities, the simulator supports a flat ground plane and simple geometric objects as obstacles, see the left image of Figure 10. More complex and realistic ground surfaces can be defined in terms of elevation maps or surface maps. By using the interface to the surface map data structure, one can run simulation experiments on terrain that has been traversed with the real vehicle. The right image on Figure 10, for example, depicts the simulated Smart car driving on a surface model that has automatically been constructed from real data acquired at an outdoor test site.

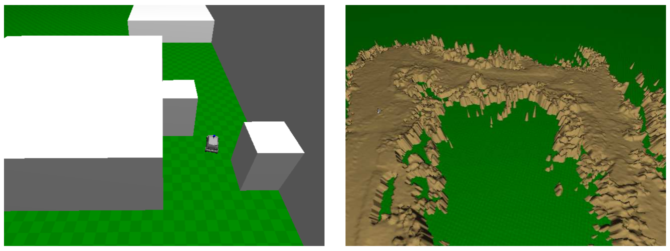
Fig. 10. The simulated environments can be composed of simple geometric objects (left) or can be automatically build from 3d laser range finder data gathered by the real vehicle (right).

B. Experiences with the Simulator

There are often controversial discussions, whether the achievements possible through simulation are worth the efforts necessary for developing and maintaining the simulator itself. From our experience, one should not spend an excessive amount of time on optimizing system parameters in simulation since the real system typically differs substantially in these aspects. On the other hand, the development of the 3d mapping capabilities, the navigation system, and especially the combination of both in one control loop was clearly facilitated by the simulation system.

For the 3d mapping algorithms, the main benefits lay in the possibility to set up simple geometric environments and to compare the mapping results with the known ground truth. Additionally, by simulating a moving vehicle with its physical properties we were able to get an intuition on how accurate the localization system has to be for achieving dense and accurate maps. It was also relatively easy to compare the results for different sensor placements, configurations, and data rates.

For the development of the navigation system, the availability of simple geometric environments was less important as this could be simulated more easily and faster by a simple planar simulation directly build into the navigation module. Far more important was the possibility to test the navigation algorithm in a dynamic setting together with the real local traversability maps calculated online. This could neither have been achieved by replaying real log files nor by using less realistic simulation.

VIII. E

A. Localization

Our approach to localization has been extensively tested and proved to be accurate and reliable in an urban environment. A typical result obtained during the validation phase is depicted in Fig. 11. The figure represents the estimated trajectory of the car overlayed on the ortho-photo of the EPFL campus. During the experiment, the car drove on areas were GPS was not available or of bad quality (close to buildings, underground, along narrow alleys bordered with trees, etc.). However, the localization algorithm was able to cope with GPS faults and provided accurate positioning estimation, such as depicted in Fig. 12.

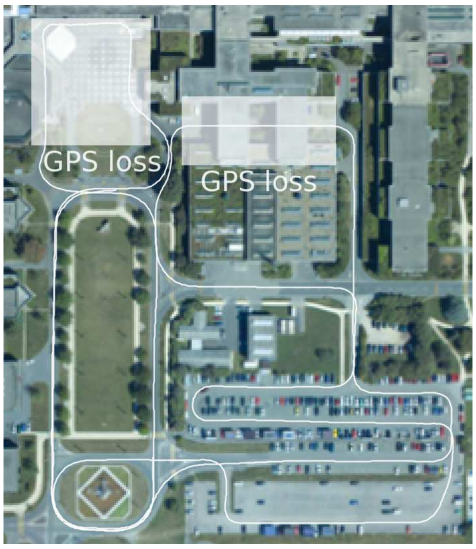

Fig. 11. Overlay of the estimated trajectory on the ortho-photo of EPFL. The areas where the GPS was not available are highlighted. The total travelled distance was 2350*m*.

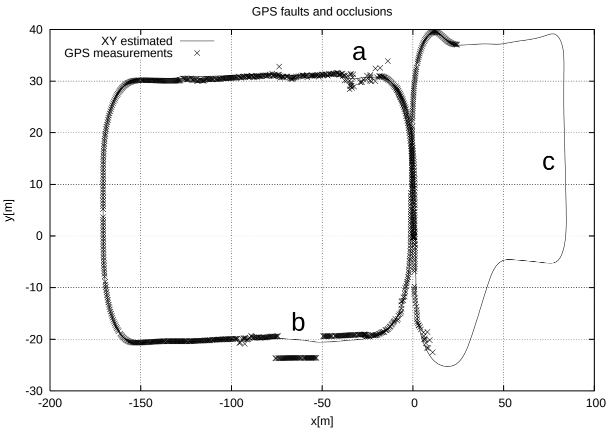

Fig. 12. The graph represents a part of the trajectory depicted in Fig. 11. In this urban environment, the GPS signal is disturbed by many objects (trees, buildings, etc.) and GPS faults are of high amplitude. The localization algorithm was able to reject erroneous GPS fixes and to provide accurate estimations. The labels a,b and c mark areas where GPS is of very poor quality (a, b) or unavailable (c).

The uncertainty associated to the pose estimation mainly depends on the GPS fixes quality. As depicted in Fig. 13, the standard deviation is low when differential GPS is available ($\sim$ 3 cm) but increases as soon as fixes become unavailable (up to 60 cm).

B. Mapping

To acquire the data, we steered the robotic car over an military test site. On its path, the robot encountered three nested loops. The goal of these experiments is to demonstrate that our representation yields a significant reduction of the memory requirements compared to a point cloud representation, while

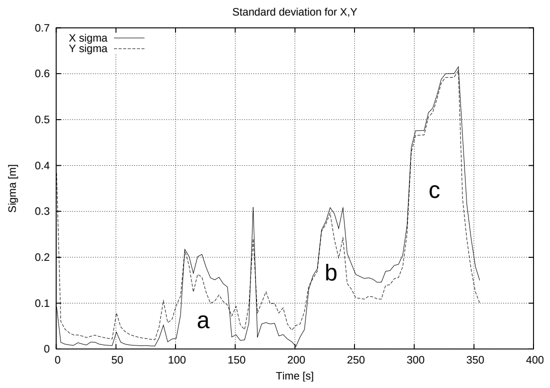

Fig. 13. Standard deviation along the north (x) and west axis (y) for the trajectory depicted in Fig. 12. The standard deviation increases when GPS quality is poor and decreases as soon as it gets better. The labels a,b and c corresponds to the regions marked in Fig. 12

still providing sufficient accuracy. Additionally, they show that our representation is well-suited for global pose estimation and loop closure. Furthermore we show that our representation is easy to apply to our simulator.

In the experiment we acquired 312 local point clouds consisting of 22,700,000 data points. The area scanned by the robot spans approximately 250 by 200 meters. During the data acquisition, the robot traversed three nested loops with a length of approximately 1,200m. Figure 14 shows a top view of the resulting MLS map with a cell size of 50cm x 50cm and 3 cutouts with a visualized smart. The yellow/light grey surface patches are classified as traversable. It requires 17.15 MBytes to store the computed map, where 36% of 200,300 cells are occupied. Compared to this the storage of the 22,700,000 data points requires 544,8 Mbytes.

IX. C

In this paper, we presented our approach to mapping of large-scale areas using an autonomous car. We first described our modified Smart car and setup. Then, we presented our approach to localization which is based on an information filter that merges the information obtained by a variety of different sensors. We furthermore presented our compact map model that is suitable to model outdoor environment in an appropriate way. We showed how to construct a model given a set of smaller map build on the fly. We describe our approach to consistently merge the individual maps into a global representation. Our approach has been implemented and tested using a real car equipped with different types of sensors. All experiments presented in this paper show the result of real world data obtained with this robot.

A

This work has partly been supported by EC under contract number FP6-IST-027140-BACS, FP6-004250-CoSy, and FP6-2005-IST-5-muFly as well as by the German Research Foundation (DFG) within the Research Training Group 1103 and under contract number SFB/TR-8. The authors would like to thank Mike Montemerlo for providing extremely helpful hints in the context of the navigation module.

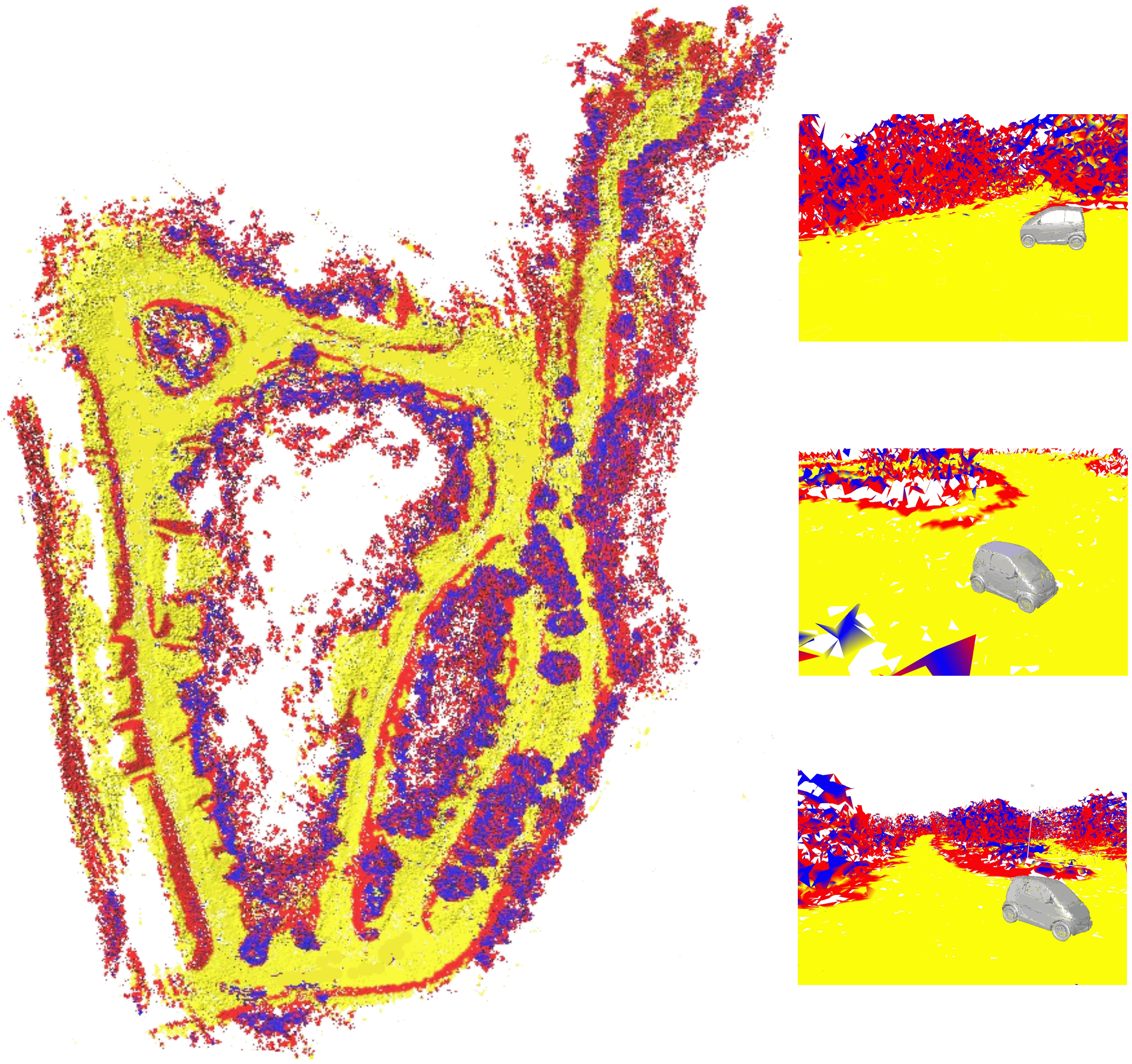

Fig. 14. The left handed image shows a top view of the resulting MLS map of a military test site with a cell size of 50cm x 50cm. The area scanned by the robot spans approximately 250 by 200 meters. During the data acquisition, the robot traversed three nested loops with a length of approximately 1,200m. On the right hand side three cutouts with a visualized smart are depicted. The yellow/light grey surface patches are classified as traversable.

R

[1] P. Allen, I. Stamos, A. Gueorguiev, E. Gold, and P. Blaer. Avenue: Automated site modeling in urban environments. In *Proc. of the 3rd Conference on Digital Imaging and Modeling*, pages 357–364, 2001.

[2] J. Bares, M. Hebert, T. Kanade, E. Krotkov, T. Mitchell, R. Simmons, and W. R. L. Whittaker. Ambler: An autonomous rover for planetary exploration. *IEEE Computer Society Press*, 22(6):18–22, 1989.

[3] L.B. Cremean, T.B. Foote, J.H. Gillula, G.H. Hines, D. Kogan, K.L. Kriechbaum, J.C. Lamb, J. Leibs, L. Lindzey, C.E. Rasmussen, A.D. Stewart, J.W. Burdick, and R.M. Murray. Alice: An information-rich autonomous vehicle for high-speed desert navigation. *Journal of Field Robotics*, 2006. Submitted for publication.

[4] DARPA. Darpa gran challenge rulebook. Website, 2004. http://www.darpa.mil/grandchallenge05/Rules 8oct04.pdf.

[5] G. Dissanayake, P. Newman, S. Clark, H.F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localisation and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, 17(3):229–241, 2001.

[6] G. Dissanayake, S. Sukkarieh, and H. Durrant-Whyte. The aiding of a low-cost strapdown inertial measurement unit using vehicle model constraints for land vehicle applications. *IEEE Transactions on Robotics and Automation*, 17(5), 2001.

[7] A. Eliazar and R. Parr. DP-SLAM: Fast, robust simultainous localization and mapping without predetermined landmarks. In *Proc. of the Int. Conf. on Artificial Intelligence (IJCAI)*, pages 1135–1142, Acapulco, Mexico, 2003.

[8] R. Eustice, H. Singh, and J.J. Leonard. Exactly sparse delayed-state filters. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 2428–2435, Barcelona, Spain, 2005.

[9] S. Fleury, M. Herrb, and F. Ingrand. GenoM. http://softs.laas.fr/openrobots/tools/genom.php.

[10] C. Früh and A. Zakhor. An automated method for large-scale, ground-based city model acquisition. *International Journal of Computer Vision*, 60:5–24, 2004.

[11] G. Grisetti, C. Stachniss, and W. Burgard. Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 2443–2448, Barcelona, Spain, 2005.

[12] J. Guivant and E. Nebot. Optimization of the simultaneous localization and map building algorithm for real time implementation. *IEEE Transactions on Robotics and Automation*, 17(3):242–257, 2001.

[13] D. Hähnel, W. Burgard, and S. Thrun. Learning compact 3d models of indoor and outdoor environments with a mobile robot. *Robotics and Autonomous Systems*, 44(1):15–27, 2003.

[14] M. Hebert, C. Caillas, E. Krotkov, I.S. Kweon, and T. Kanade. Terrain mapping for a roving planetary explorer. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 997–1002, 1989.

[15] A. Kelly and A. Stentz. Rough terrain autonomous mobility, part 1: A theoretical analysis of requirements. *Journal of Autonomous Robots*, 5:129–161, 1998.

[16] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. Technical report, USC Center for Robotics and Embedded Systems, CRES-04-002, 2004.

[17] P. Kohlhepp, M. Walther, and P. Steinhaus. Schritthaltende 3D-Kartierung und Lokalisierung für mobile inspektionsroboter. In *18. Fachgespräche AMS*, 2003. In German.

[18] S. Lacroix, A. Mallet, D. Bonnafous, G. Bauzil, S. Fleury and; M. Herrb, and R. Chatila. Autonomous rover navigation on unknown terrains: Functions and integration. *Int. Journal of Robotics Research*, 21(10-11):917–942, 2002.

[19] J.J. Leonard and H.F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(4), 1991.

[20] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The digital michelangelo project: 3D scanning of large statues. In *Proc. SIGGRAPH*, pages 131–144, 2000.

[21] Y. Liu, R. Emery, D. Chakrabarti, W. Burgard, and S. Thrun. Using EM to learn 3D models with mobile robots. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2001.

[22] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Journal of Autonomous Robots*, 4:333–349, 1997.

[23] C. Martin and S. Thrun. Online acquisition of compact volumetric maps with mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, Washington, DC, 2002. ICRA.

[24] M. Montemerlo, S. Thrun D. Koller, and B. Wegbreit. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Proc. of the Int. Conf. on Artificial Intelligence (IJCAI)*, pages 1151–1156, Acapulco, Mexico, 2003.

[25] M. Montemerlo, N. Roy, S. Thrun, D. Hähnel, C. Stachniss, and J. Glover. CARMEN – the carnegie mellon robot navigation toolkit. http://carmen.sourceforge.net, 2002.

[26] M. Montemerlo and S. Thrun. A multi-resolution pyramid for outdoor robot terrain perception. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2004.

[27] H.P. Moravec. Robot spatial perception by stereoscopic vision and 3d evidence grids. Technical Report CMU-RI-TR-96-34, Carnegie Mellon University, Robotics Institute, 1996.

[28] A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann. 6d SLAM with approximate data association. In *Proc. of the 12th Int. Conference on Advanced Robotics (ICAR)*, pages 242–249, 2005.

[29] C.F. Olson. Probabilistic self-localization for mobile robots. *IEEE Transactions on Robotics and Automation*, 16(1):55–66, 2000.

[30] E. Olson, J. Leonard, and S. Teller. Fast iterative alignment of pose graphs with poor estimates. In *ICRA*, 2006.

[31] C. Parra, R. Murrieta-Cid, M. Devy, and M. Briot. 3-d modelling and robot localization from visual and range data in natural scenes. In *1st International Conference on Computer Vision Systems (ICVS)*, number 1542 in LNCS, pages 450–468, 1999.

[32] K. Pervölz, A. Nüchter, H. Surmann, and J. Hertzberg. Automatic reconstruction of colored 3d models. In *Proc. Robotik*, 2004.

[33] P. Pfaff and W. Burgard. An efficient extension of elevation maps for outdoor terrain mapping. In *In Proc. of the Int. Conf. on Field and Service Robotics (FSR)*, pages 165–176, 2005.

[34] Hanan Samet. *Applications of Spatial Data Structures*. Addison-Wesley Publishing Inc., 1989.

[35] R. Simmons. IPC – inter process communication. http://www.cs.cmu.edu/afs/cs/project/TCA/www/ipc/index.html.

[36] S. Singh and A. Kelly. Robot planning in the space of feasible actions: Two examples. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1996.

[37] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial realtionships in robotics. In I. Cox and G. Wilfong, editors, *Autonomous Robot Vehicles*, pages 167–193. Springer Verlag, 1990.

[38] Russell Smith. Open dynamics engine. Website, 2002. `http://www.q12.org/ode/ode.html`.

[39] S. Sukkarieh, E.M. Nebot, and H. Durrant-Whyte. A high integrity imu/gps navigation loop for autonomous land vehicle application. *IEEE Transactions on Robotics and Automation*, 15(3), 1999.

[40] S. Thrun. An online mapping algorithm for teams of mobile robots. *Int. Journal of Robotics Research*, 20(5):335–363, 2001.

[41] S. Thrun, W. Burgard, and D. Fox. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping. In *Proceedings of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, San Francisco, CA, 2000. IEEE.

[42] S. Thrun, D. Hähnel, D. Ferguson, M. Montemerlo, R. Triebel, W. Burgard, C. Baker, Z. Omohundro, S. Thayer, and W. Whittaker. A system for volumetric robotic mapping of abandoned mines. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, Taipei, Taiwan, 2003.

[43] S. Thrun, Y. Liu, D. Koller, A.Y. Ng, Z. Ghahramani, and H. Durrant-Whyte. Simultaneous localization and mapping with sparse extended information filters. *Int. Journal of Robotics Research*, 23(7-8):693–704, 2004.

[44] S. Thrun, C. Martin, Y. Liu, D. Hähnel, R. Emery Montemerlo, C. Deepayan, and W. Burgard. A real-time expectation maximization algorithm for acquiring multi-planar maps of indoor environments with mobile robots. *IEEE Transactions on Robotics and Automation*, 20(3):433–442, 2003.

[45] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. Winning the darpa grand challenge. *Journal of Field Robotics*, 2006. To appear.

[46] R. Triebel, F. Dellaert, and W. Burgard. Using hierarchical EM to extract planes from 3d range scans. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2005.

[47] R. Triebel, P. Pfaff, and W. Burgard. Multi-level surface maps for outdoor terrain mapping and loop closing. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2006.

[48] C. Urmson. *Navigation Regimes for Off-Road Autonomy*. PhD thesis, Robotics Institute, Carnegie Mellon University, 2005.

[49] Carl Wellington, Aaron Courville, and Anthony Stentz. Interacting markov random fields for simultaneous terrain modeling and obstacle detection. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2005.

[50] O. Wulf, K-A. Arras, H.I. Christensen, and B. Wagner. 2d mapping of cluttered indoor environments by means of 3d perception. In *ICRA-04*, pages 4204–4209, New Orleans, apr 2004. IEEE.

[51] C. Ye and J. Borenstein. A new terrain mapping method for mobile robot obstacle negotiation. In *Proc. of the UGV Technology Conference at the 2002 SPIE AeroSense Symposium*, 1994.