The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

Real-time Object Classification in 3D Point Clouds Using Point Feature Histograms

M. Himmelsbach, T. Luettel and H.-J. Wuensche

Abstract— **This paper describes a LIDAR-based perception system for ground robot mobility, consisting of 3D object detection, classification and tracking. The presented system was demonstrated on-board our autonomous ground vehicle MuCAR-3, enabling it to safely navigate in urban traffic-like scenarios as well as in off-road convoy scenarios. The efficiency of our approach stems from the unique combination of 2D and 3D data processing techniques. Whereas fast segmentation of point clouds into objects is done in a $2\frac{1}{2}$D occupancy grid, classifying the objects is done on raw 3D point clouds. For fast object feature extraction, we advocate the use of statistics of local point cloud properties, captured by histograms over point features. In contrast to most existing work on 3D point cloud classification, where real-time operation is often impossible, this combination allows our system to perform in real-time at 0.1s frame-rate.**

I. INTRODUCTION

In this paper we address the problem of segmenting 3D scan data into objects of known classes. Given the set of 3D points acquired by a range scanner, the goal of segmentation is to attribute the points to a set of candidate object classes. In the context of ground robot mobility, this segmentation capability is not only essential for high-level tasks like scene understanding and planning, but can also be used for scan registration and robot localization, e.g. in a SLAM framework [1]. Besides, knowing the object's class is especially useful in dynamic environments, both for planning and estimation: estimation can be improved by making use of appropriate dynamic models, and planning can incorporate knowledge about the behavior or intentions typical of a certain object class.

Our approach to perception is decomposed into three main steps: segmentation, classification and tracking. The segmentation step is performed on an occupancy grid, yielding connected components of grid cells not belonging to the ground surface. In an efficient operation we determine all the 3D LIDAR point measurements corresponding to the segmented objects. In the classification step we extract features from an object's point cloud, capturing the distribution of local spatial and reflectivity properties extracted over a fixed-size support volume around each point. In a supervised learning framework, a support vector machine (SVM) classifier is trained to discriminate the classes of interest, e.g. other traffic participants in our case, given hand-labeled examples of point clouds.

All authors are with department of Aerospace Engineering, Autonomous Systems Technology (TAS), University of the Bundeswehr Munich, Neubiberg, Germany.
Contact author email: `michael.himmelsbach@unibw.de`

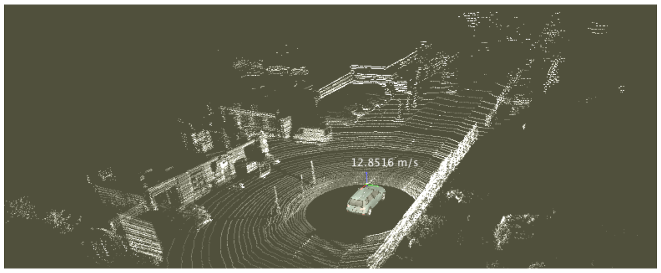

Fig. 1. Inertially corrected cloud of 10^5 3D points for one revolution (0.1s) of the Velodyne LIDAR, mounted on the roof of MuCAR-3. Note the different scales of gray, corresponding to the intensity of the reflected beam. *All figures are best viewed in color.*

The method is not restricted to a particular robot or sensor, however we describe and demonstrate it using our vehicle MuCAR-3 (Munich Cognitive Autonomous Robot Car, 3^{rd} generation), a VW Touareg equipped with a Velodyne HDL-64 LIDAR (see fig. 1).

A. Related Work

With range scanning devices becoming standard equipment in mobile robotics, the task of 3D scan segmentation and classification is one of increasing practical relevance. Interestingly, although range scanners were the primary sensor at the DARPA Urban Challenge 2007, segmentation was primarily done on $2\frac{1}{2}$-D occupancy grids. If at all, classification of segmented objects was done in the 2D domain, by fitting L-shapes or bounding boxes and verifying them against simple rules [2], [3]. Classification was probably omitted because of the strict rules of the competition, that ensured that every object detected within the road boundaries could only correspond to another vehicle or static obstacles which had to be avoided without having to classify them

In contrast, both Anguelov *et. al.* [4] and Lalonde *et. al.* [5] describe methods where every single point of a scan is assigned a class label. Given a labeled point cloud, segmenting the scan is then straight-forward. While the features extracted for each point do not differ considerably – both methods use local point cloud statistics for feature extraction, to be detailed later –, different classification paradigms are followed. Anguelov *et. al.* [4] model a point's class label by a probability distribution conditioned on the local features and the labels in the point's neighborhood. They thus enforce spatial contiguity, exploiting the fact that adjacent points in the scan should have similar labels. This distribution is modeled by a Markov Random Field (MRF), whose parameters are determined in a supervised learning stage such that the resulting classifier maximizes the margin between the classes learned, like SVMs do. Although no timing results are given in [4], it can be concluded from [6] that the method does not permit real-time use.

Lalonde *et.al.* [5] learn a parametric model of the feature distribution for each class by fitting a Gaussian mixture model (GMM) using the Expectation Maximization (EM) algorithm on a hand labeled training data set. Spatial contiguity is accounted for by running simple rule-based filters after classification, e.g. by changing a point's label to the most frequent class among its neighbors. However, to make their method perform in real-time, some modifications are necessary. Especially, they no longer classify individual points, but an artificial prototype point of all points contained in a 3D voxel grid cell, such that 7000 voxels/sec. can be classified.

We take a quite different, unique approach to object classification in 3D point clouds, in that segmentation is based on the compressed data contained in a $2\frac{1}{2}$D occupancy grid. We then make use of the rich information contained in the Velodyne's 3D point clouds by again switching the domain to 3D, now classifying only subsets of the scan's total point cloud, with evidence that each subset represents an individual object. To get a compact description of an object for classification, we build histograms over features computed for individual points of an object. Thanks to the efficient combination of 2D and 3D data processing techniques, classification of objects represented by their 3D point clouds is possible in real-time on-board an autonomous vehicle.

II. OBJECT DETECTION

A. Occupancy Grid Mapping

We use a $2\frac{1}{2}$-D ego-centered occupancy grid of dimension $100\text{m}\times 100\text{m}$, each cell covering a small ground patch of $0.15\text{m}\times 0.15\text{m}$. Each cell stores a single value expressing the degree of how occupied that cell is by an obstacle. In our implementation this value is a metric length with the physical unit [m]. Before we detail its meaning and calculation, note that in our approach we create a new occupancy grid on each new LIDAR revolution, i.e. every 0.1s. Thus, we do not accumulate data for a longer time. The reasons for this decision are twofold. First, one revolution of the Velodyne supplies about 10^5 3D points, which proved to be sufficient. Second, the quality of an accumulated occupancy grid can easily deteriorate if the physical movement of the sensor is not estimated with very high precision. Small angular deviations in the estimate of the sensor's pose can result in large errors. Registering scans against each other, e.g. using the ICP algorithm [7] or some of its derivatives, could solve this problem, but would require substantial additional computational load.

For calculating the occupancy values, we first inertially correct the LIDAR scan, taking the vehicle's motion into account (exploiting IMU and odometric information). This is done by simultaneously moving the coordinate system of the vehicle while transforming the local LIDAR measurements to global 3D space. After a frame is completed, all points are transformed back into the last local coordinate system of the vehicle, simulating a scan as if all measurements were taken at a single point of time instead of the 0.1s time period of one LIDAR revolution.

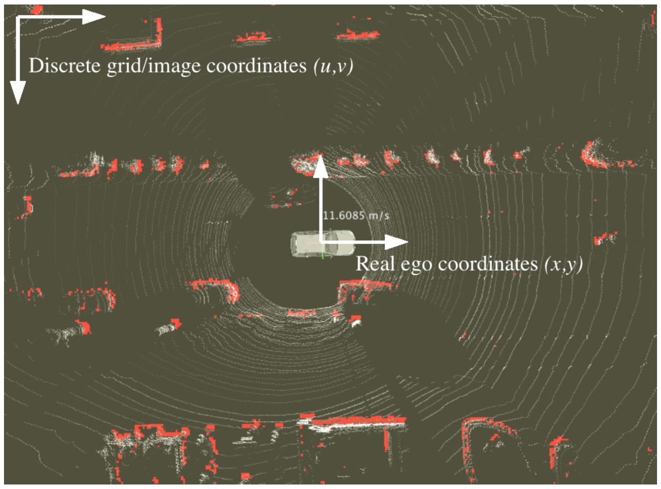
Discrete grid/image coordinates (u,v)
11.6085 m/s
Real ego coordinates (x,y)

Fig. 2. Occupancy grid (with only profoundly occupied cells shown in *red*) and superimposed point cloud. The geometric relation between discrete grid coordinates and real ego coordinates remains static.

Similar to Thrun *et. al.* [8], each cell's value is then calculated to be the maximum absolute difference in z-coordinates of all points falling into the respective grid cell. When a grid cell is hit by a laser beam and its occupancy value is updated, we store the laser read at the cell such that it can be queried for later processing, to be detailed in Subsec. II-C. Fig. 2 shows the occupancy grid with superimposed point cloud.

B. Object Hypotheses from Segmentation

To get initial object hypotheses, we next perform a segmentation of the occupied grid cells by finding connected components of grid cells. In order to apply the connected components algorithm, the grid first needs to be binarized. This is achieved by simply thresholding the occupancy values of all cells against a suitable value, setting all cells below the threshold to zero and all others to one. Note that segmentation results are greatly influenced by the choice of the threshold: a value too large may result in adjacent objects being detected as one, whereas setting it too small may cause one object to be split into pieces. By careful investigation, we choose the threshold to be 0.15m, and these problems rarely occured. Then, standard connected component algorithms known from machine vision [9] can be applied, that assign each grid cell c_i the label $label_i$ of the connected component it belongs to.

For each connected component cc_k, we formulate an object hypothesis of unknown class, represented as a 3D bounding box. The x- and y-axis of the object's bounding box can be calculated from the discrete grid coordinates $gc_i = (u, v)^T$ of all cells c_i belonging to the respective

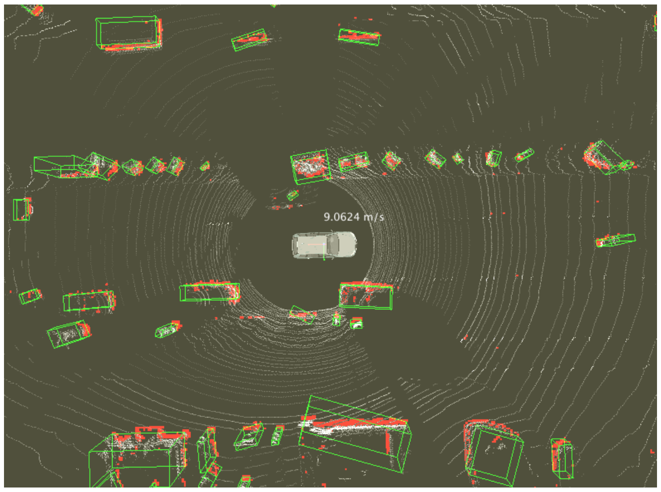

Fig. 3. Occupancy grid with objects detected by segmentation, represented by 3D bounding boxes (*green*).

connected component, $cc_k = \{gc_i^{ego} | label_i = k\}$. Here, the "ego" superscript is used to denote that all grid coordinates are now expressed in the ego coordinate system, a conversion greatly simplified by the static geometric relation between the ego and the grid cells. The axes then correspond to the orthonormal eigenvectors e_1, e_2 of the coordinates' covariance matrix Σ_{cc_k}, sorted in descending order w.r.t. to the corresponding eigenvalues, i.e. $d_1 \geq d_2$.

The boxes' dimensions in the plane are found by linearly transforming all $gc_i^{ego} \in cc_k$ into the coordinate system defined by the eigenvectors, $gc_i^{ego^*} = (e_1|e_2)gc_i^{ego}$, and taking the extremes over the resulting coordinates. The position pos_k of the object hypothesis is simply the center of gravity of the connected component, i.e. $pos_k = |cc_k|^{-1}\Sigma(gc_i^{ego} \in cc_k)$. While these boxes are not optimal in terms of minimum area, they are assured to enclose all grid cells of the connected component.

Lacking knowledge about the scene's ground plane, we make the assumption that the object's z-axis is orthogonal to the ego's xy-plane and set its z-dimension to the maximum z-coordinate of all cells part of the connected component, thus obtaining the final 3D bounding box object hypothesis.

Fig. 3 shows the result of applying the outlined object detection algorithm to the occupancy grid shown in fig. 2.

C. *Object Point Clouds*

As mentioned earlier, we want to classify the detected objects based on their 3D point measurements. However, segmentation just provides us with a bounding box representation of an object. Remember that when updating a grid cell with a laser read, we store the laser read at the cell[1]. The naive approach to obtain the point cloud corresponding to an object would thus be to simply collect all laser reads stored at the respective connected component. This, however, is not expedient in our case. The reason is that connected components are only made of cells for which a certain z-coordinate difference was observed, and it can not be taken for granted that all measurements of an object fall into such cells. Thus, taking only the points from the connected component cells would probably miss a large number of object measurements and harden the following classification step.

[1]In fact, the occupancy grid is implemented general enough to allow storage of any kind of data at the cells. For this application, however, storing laser reads is appropriate, and we use the terms "data" and "laser read(s)" interchangeably.

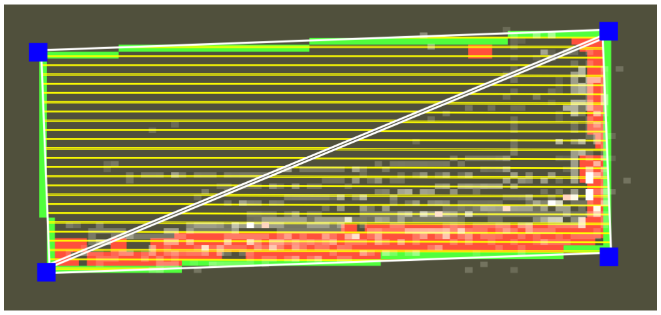

Fig. 4. Querying data from the occupancy grid, with the query formulated as a polygon. The polygon (with vertices shown *blue*) gets split into triangles (2 in this example, *white*) and scan conversion is issued on all triangles (*yellow* scan lines). The query is answered by returning all data within the scanned grid cells (i.e. the laser reads, shown *grey*). Note that not all laser reads fall into cells part of the connected component (*red*).

Instead, we would like to extract all laser reads contained in the object's 3D bounding box. To do so, we augment the grid with a facility to answer queries for data formulated as arbitrary (convex and non-convex) polygons, with vertices defined in the ego coordinate system. E.g., given as a query the polygon $Q = \{v_1^{ego}, .., v_N^{ego}\}$, the grid will answer by returning the union of data stored at all cells falling within the boundaries of polygon Q. Given such a query, we first transform the vertex coordinates v_i from ego to grid coordinates and split the resulting polygon into triangles. We next issue the triangle scan conversion algorithm on each resulting triangle, such that every grid cell contained in the original polygon formulation will be visited once. Answering the query is then a simple matter of collecting all the laser reads stored at the visited cells.

This is illustrated in fig. 4 for the case of extracting laser reads for an object hypothesis, where the query is given in terms of a polygon representation of the bottom plane of the object's 3D bounding box. Note that, as expected, not all laser reads fall into grid cells that are part of the corresponding connected component.

III. CLASSIFICATION USING POINT FEATURE HISTOGRAMS

In this section we detail the features extracted from an object's point cloud to be used for classification. Literature on extracting features local to a single point is vast. Among these, 3D Shape Contexts [10] and Spin Images [11] can be considered the most famous. While these features proved well suited for the task of object recognition, their descriptive power comes at the cost of an increased dimensionality, ranging from 200 up to 1000 feature values for a single point, and according computational needs. As we are targeting real-time performance, we have to resort to much simpler features, such as the ones used by Anguelov *et.al.* [4] or Lalonde *et. al.* [5].

No matter what point features we choose, none of them can be applied directly, as in our case the extracted features must provide a compact description of a cloud of possibly many points, whereas the features described above only need to describe prominent properties of single points. This issue is addressed by introducing histograms over point features computed for individual points of an object. On the other hand, the features that can be computed from point clouds are not restricted to local point properties. Instead, some of the features should also capture global object properties, like the object's dimensions or volume etc.. Hence, we will use both local and global features to describe point clouds.

Formally, a point cloud P can be written as $P = \{l_1, ..., l_M\}$, where $l_i = \{x_i, y_i, z_i, I_i\}$ denotes a single laser read, consisting of the coordinates x_i, y_i, z_i of the measured 3D point and the intensity $I_i \in [0, 255]$ of the reflected beam. For real data, the size of an object's point cloud typically ranges from $M = 100...1000$. However, with objects closer to the sensor, point clouds of $M = 10000$ points are possible in the extreme. Computing point features for every laser read is intractable for such a large number of points when targeting real-time operation. We thus perform uniform down-sampling of each object's point cloud to reduce the number of points to a constant of $M := 200$ prior to feature extraction.

We now describe the features extracted from these reduced point clouds in detail.

A. Object Level Features

We call features not involving any local computations "object level feature". We include four of them in our final feature vector, all of which are scalar valued:

- Maximum object intensity $I_{max} = \max I_i$
- Mean object intensity $\mu_I = M^{-1}\Sigma_i I_i$
- Object intensity variance $\sigma_I = \Sigma_i(I_i - \mu_I)^2$
- Object volume V, computed from the corresponding 3D bounding box

Obviously, $i = 1...M$ in all the above computations. Summarizing, these features give an idea about an object's extent and the reflectance properties of its surface material.

B. Histograms Over Point Level Features

Whereas object level features do not involve local point properties, we now turn to the types of features capturing the statistics of local point cloud properties.

To transfer the features from the point level to the object level, we introduce a histogram for every point feature and update the feature's histogram with the evaluation of the feature at every single point. After a feature has been computed for all points, we normalize the corresponding histogram by dividing every bin value by M. We then include the bins of the resulting histograms into our final feature vector. To be able to define the histogram bins over a fixed finite range, we require that all point features be normalized to only take values in the range $0...1$.

- Lalonde features L_1, L_2 and L_3
  Lalonde *et. al.* [5] compute local point features expressing the *scatter-ness* (L_1), *linear-ness* (L_2) and *surface-ness* (L_3) at a point by inspecting the distribution of neighboring points. To compute the features, they perform an eigenvalue analysis of the covariance matrix of the neighboring points' 3D coordinates, yielding eigenvectors e_1, e_2, e_3 with sorted eigenvalues $d_1 \geq d_2 \geq d_3$. They then set $L_1 = d_1$, $L_2 = d_1 - d_2$ and $L_3 = d_2 - d_3$. As there is no practical upper bound to any of these features, we make the substitution $d_i \mapsto \frac{d_i}{\Sigma_i d_i}$ such that $\forall h_i : 0 \leq h_i \leq 1$, as required[2]. For transforming these point-level features to the object-level, we add 3 histograms over the range $[0; 1]$ to the final feature vector, one for each feature L_i.
- Anguelov feature A_1
  Anguelov *et. al.* [4] describe two features, but only one is used in our work as the other does not differ significantly from the ones calculated above. The feature we take defines a vertical cylinder of height 1m and radius 0.25m around the point the feature is computed for. This cylinder is then vertically divided into 5 parts $A_{1,i}$ of equal size, and each is assigned the fraction of all points in the cylinder falling into it. Thus, if the point was located on a vertical plane, all bins would be assigned approx. equal fractions. This adds another five histograms, one for each of the cylinder subdivisions $A_{1,i}$, to the final feature vector, capturing the distribution of A_1 in the given object point cloud.

We haven't yet given a precise definition of the term "point neighborhood" used in the above computations. This refers to the point's 20 nearest neighbors within a fixed-bound radius of $0.5m$. These are efficiently found by constructing a *k*D-tree from the object's point cloud once and doing the nearest neighbor searches in this tree.

C. Training the SVM classifier

For classifying objects, next a support vector machine (SVM) classifier is trained on the hand-labeled training data set. Like the maximum margin MRFs (M^3) used by Anguelov *et. al.*, the SVM also maximizes the margin between the different classes it is trained on, but lacks the concept of spatial contiguity. However, segmentation is done prior to classification in our approach, rendering the spatial contiguity property less important. We use the common ν-SVM variant, that allows for some mislabeled examples in case the classes are not completely separable in feature space [12].

The operation of the ν-SVM depends on two parameters: C is a penalty parameter for weighting classification errors and γ is a kernel function parameter. To also determine

[2]This is possible as the covariance matrix is positive semi-definite, hence all its eigenvalues are nonnegative.

the optimal choice of these parameters, we perform a grid-search in a suitable subspace of parameter values. At each grid resolution, the classification performance for different pairings of C, γ, given by the grid cells, is evaluated by randomly splitting the training data set into two folds of equal size. Applying the concept of cross-validation, one fold is then used for training the SVM using the current parameter choices and the other fold for evaluation. The search then iterates by refining the resolution of the grid and centering it at the best parameter choices of the last iteration.

IV. Choosing a Size for Point Feature Histograms

A final important parameter in object feature extraction is the number of bins in each of the histograms. One possible choice is to choose the number of bins b for all histograms involved such that the overall margin in feature space between examples of different labels is maximized.

The work of Bachrach *et.al.* [13] on margin based feature selection provides us with the necessary concepts for doing so. In their work, every feature is assigned an appropriate weight so that the margin is maximized. Then, features that are assigned large weights are selected while those with low weights are discarded. Given such weights w, the margin for one example with feature vector f is defined as

$$m_F^w(f) = 0.5\,(||f - nearmiss(f)||_w - ||f - nearhit(f)||_w) \quad (1)$$

where $||x||_w = \sqrt{\sum_i w_i^2 x_i^2}$ and $nearmiss(f)$ and $nearhit(f)$ denote the examples in F closest to f in feature space of same and different label than f. The margin for the complete labeled data F then simply is

$$M^w(F) = \sum_{f \in F} m_{F/f}^w(f) \quad (2)$$

Bacharach *et.al.* also show how to find the weights maximizing M_F^w with an iterative algorithm called Simba [13].

We can use Simba to decide for the number of bins in our histograms by running it multiple times, altering the number of bins of the histograms used for extracting the features f in each run. We may then select the number of bins for our histograms as those resulting in the largest margin in our training data.

We ran Simba for histogram sizes of $b = 2..10$ for 500 iterations each time. We did not run it for larger histogram sizes as the resulting feature vector size would grow too large to still allow real-time execution. Fig. 5 shows the resulting margins for different choices of b. According to these results, we choose the histogram size as $b = 7$. An interesting point to note is that the bins of the histograms were assigned approximately equally large weights by Simba, so none of the bins would be discarded by max-margin feature selection.

With regard to the final feature vector, we have four scalar object-level features. In addition, we have eight histograms over point-level features, each contributing $b = 7$ bins. The final feature vector thus has dimension 60 and takes the form $f = (I_{max}, \mu_I, \sigma_I, V, H_{L_1}^7, H_{L_2}^7, H_{L_3}^7, H_{A_{1,1}}^7, ..., H_{A_{1,5}}^7)$,

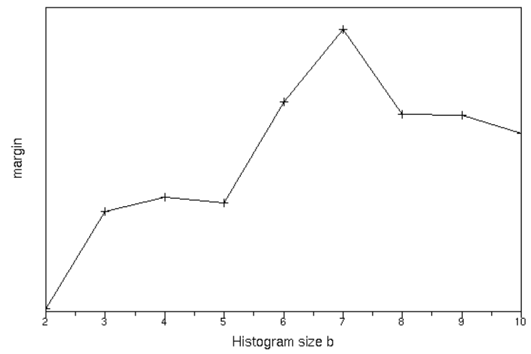

Fig. 5. Margins for different histogram sizes b after running Simba on hand-labeled training data.

where H_v^b denotes the b bin values of the histogram over the scalar valued variable v.

V. RESULTS AND APPLICATIONS

A. Two-class Classification Results

This framework for object classification has been tested on the task of discriminating objects belonging to the class of passenger cars from other types of objects. Bearing in mind that the classifier will be presented features f extracted from automatically detected objects in a real application, semi-automatic data labeling was done. We ran our object detection algorithm on the scans of diverse urban and non-urban traffic scenes and visualized the detected objects in a GUI. Via simple user interaction, each detected object could be assigned one of the labels “vehicle” or “non-vehicle” (mostly vegetation and buildings). For each labeled object, we extracted the corresponding point cloud as described above and stored the points together with the label for later training the SVM.

To get an impression of how such training data looks like, fig. 6 shows a few of the extracted examples. The complete training set contained a total of 284 examples, split into 109 positive and 175 negative ones. As can be seen, the training data set contained positive examples for vehicles sensed from different viewing directions and distances. Also, comparing the intensities across both classes, separating the classes based on intensity information alone seems impossible.

We then run the SVM training procedure as described in subsec. III-C on the collected data. Note that the cross-validation result of the last grid-search iteration already expresses the accuracy of the trained classifier, where the classifier's performance is evaluated on data different from the one it was trained on. We thus report the accuracy achieved in cross-validation in the last iteration of grid-search. Here, only 6 of the 182 examples of the evaluated fold are assigned the wrong class label, resulting in an accuracy of $\frac{176}{182} \approx 96.7\%$.

Fig. 7 shows the output of the classifier on the objects detected in one LIDAR frame of an urban environment. Next, we briefly describe a real-world application built on top of the presented object classifier, demonstrating its real-time performance.

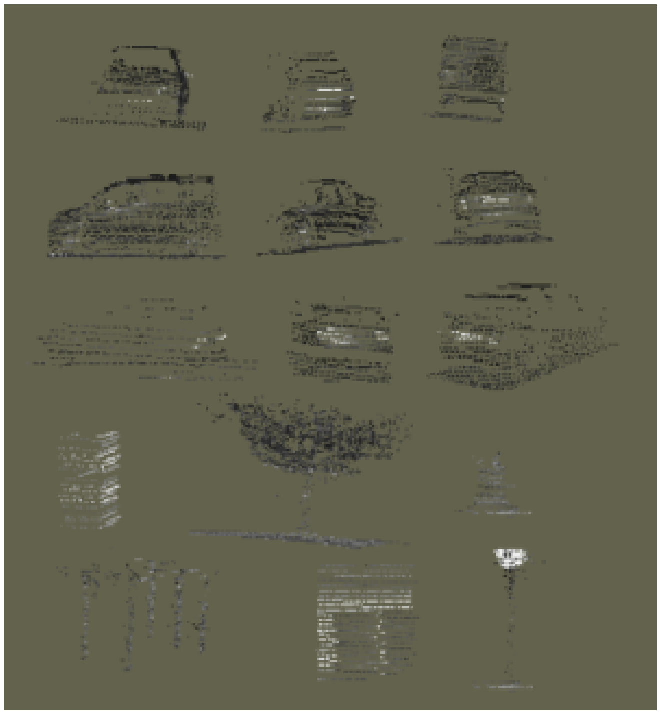

Fig. 6. Some hand-labeled examples of point clouds used for training a vehicle classifier. Positive examples (*top 3 rows*) and negative examples (*bottom 2 rows*).

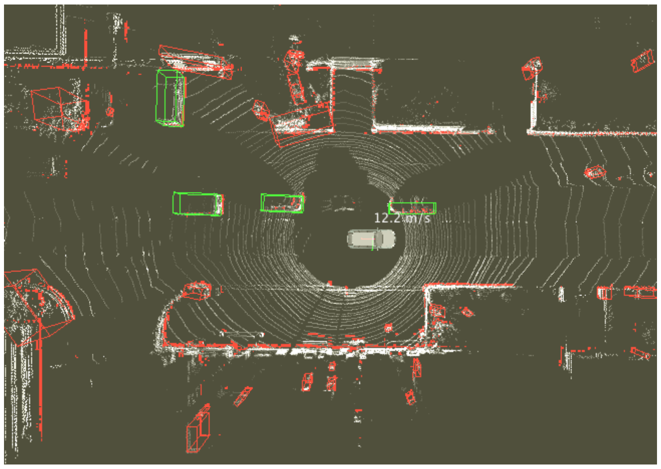

Fig. 7. Result of applying the vehicle classifier to the point clouds of objects detected in one LIDAR frame of an urban environment: vehicles (*green*) and other objects (*red*).

B. Object Tracking

To validate the presented LIDAR-based perception framework in real scenes, we integrated it into a system for object tracking in a convoy scenario. Here, MuCAR-3 is to autonomously follow the path taken by the vehicle leading the convoy. Especially, when loosing sight of the leader object, the classifier must guarantee that no other perceived object is assigned the role of the leader by fault. Instead, in this case the robot should slow down until it has again found the leader object, while navigating autonomously along the previously detected path of the leader.

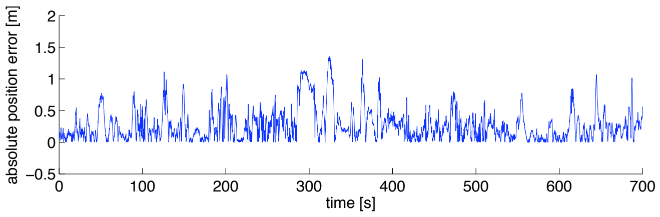

Fig. 8. Absolute position estimation error over the complete time of convoy driving. Note the largest errors appear when driving sharp curves, where the appearance of the convoy leader keeps rapidly changing (compare fig. 9).

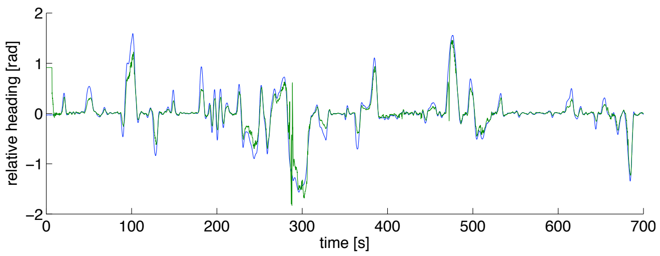

Fig. 9. Relative heading of convoy leader (*blue*) and estimates (*green*) over the complete time of convoy driving.

To achieve this, only detected objects classified as a vehicle will be considered as a convoy leader. To further improve robustness of perception and to both allow smooth longitudinal and lateral control, the convoy leader's pose and velocity is estimated in a multiple model Kalman Filtering framework[3]. At each iteration, i.e. at 10Hz, gated nearest neighbor data association is performed to select one of the possibly many vehicles as the one corresponding to the current convoy leader, and the estimates are updated accordingly.

With this system, autonomous convoy following at speeds up to 20m/s could be demonstrated for runs over distances of up to 60km, leading through different kinds of environments, including inner city, country roads, forest tracks, sharp serpentines and trails with large potholes causing violent pitch motions of the follower vehicle. Even in forests, where the many tree trunks cast the detection and classification of hundreds of objects, processing on a single core of an Intel 2.6GHz Xeon machine (including tracking and evaluating "tentacles" [16] for local obstacle avoidance) could always be finished before the arrival of the next scan, i.e. within 0.1s.

Figures 8 and 9 show some detailed results of convoy driving obtained for the scenario shown in fig. 10. A video showing a visualization of the processing going on on-board MuCAR-3 during autonomous convoy driving can be found at *http://www.unibw.de/lrt8*.

VI. CONCLUSIONS AND FUTURE WORKS

A. Conclusions

We presented a complete system for perception of 3D objects in LIDAR data. The success of the system was demonstrated in an object tracking application used for

[3]An IMM estimator [14] based on an Unscented Kalman Filter [15].

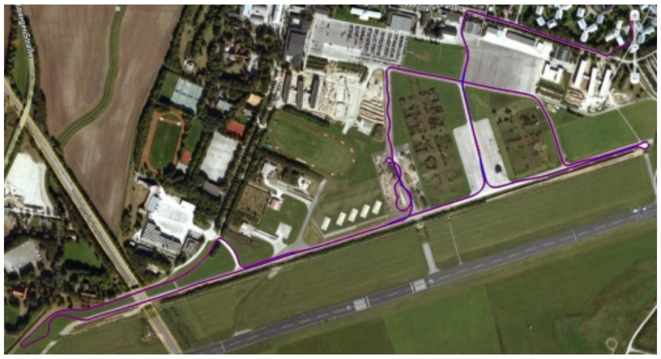

Fig. 10. Map view of the driven convoy scenario (*blue:* leader path, *red:* follower path). The total length of the track is 5937m, driven at an average speed of 6.9m/s, with 17.7m/s top speed. Both vehicles' position and heading were measured for reference by two Inertial Navigation Systems (INS) with D-GPS accuracy (*aerial image: Google Earth*).

autonomous convoy following during the Elrob 2008[4]. While segmentation and classification of objects in 3D point clouds has been done by several authors before, the way we combine 2D and 3D data processing techniques seems to be unique to our approach. While more experiments are necessary to thoroughly evaluate the potential of classifying objects in LIDAR data based on point feature histograms, the main benefit of our approach becomes obvious even at this stage of development: being able to detect, classify and track objects based on large-sized 3D point clouds, containing as much as 10^5 measurements, while still reaching real-time performance of better than 10Hz.

B. Future Works

The paper introduced some issues worth to be considered. Evidently, more object classes must be learned in order to analyze the potential of the presented classification method. This will truly show if transferring local point features to the object level is general enough to describe and discriminate various object classes. Besides, research on more types of object level features should be carried out, independent of the success of using point feature statistics for object classification.

There are a number of immediate improvements possible, at different stages of processing: For example, computing minimal-area bounding boxes could improve segmentation of the 3D point cloud. Also, the naive sampling approach for reducing the number of points in feature extraction could be augmented with models for point cloud saliency [17], sampling points of higher saliency with higher probability. It could further be investigated whether some of the parameters involved can be determined automatically from the data. For example, choosing the size of a point's neighborhood for local feature computation should depend on the distance to the point, due to the low angular resolution typical to LIDAR systems. The work of Unnikrishnan *et. al.* [18] on selecting scale from point cloud data already points into that direction.

[4]European Land-Robot Trial, *http://www.elrob.org/*

VII. ACKNOWLEDGMENTS

The authors gratefully acknowledge funding by german cluster of excellence CoTeSys, Cognition for Technical Systems (also see *http://www.cotesys.org*) and partial support by the Deutsche Forschungsgemeinschaft within the Transregional Collaborative Research Centre 28 *Cognitive Automobiles*.

REFERENCES

[1] C.-C. Wang, C. Thorpe, and S. Thrun, "Online Simultaneous Localization and Mapping with Detection and Tracking of Moving Objects: Theory and Results from a Ground Vehicle in Crowded Urban Areas," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[2] S. Kammel, J. Ziegler, B. Pitzer, M. Werling, T. Gindele, D. Jagzent, J. Schröder, M. Thuy, M. Goebl, F. von Hundelshausen, O. Pink, C. Frese, and C. Stiller, "Team AnnieWAY's autonomous system for the DARPA Urban Challenge 2007," *International Journal of Field Robotics Research*, 2008.

[3] D. Ferguson, M. Darms, C. Urmson, and S. Kolski, "Detection, Prediction, and Avoidance of Dynamic Obstacles in Urban Environments," in *Proceedings of the IEEE International Conference on Intelligent Vehicles (IV08)*, 2008, pp. 1149–1154.

[4] D. Anguelov, B. Taskar, V. Chatalbashev, D. Koller, D. Gupta, G. Heitz, and A. Ng, "Discriminative Learning of Markov Random Fields for Segmentation of 3D Scan Data," in *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 169–176.

[5] J.-F. Lalonde, N. Vandapel, D. Huber, and M. Hebert, "Natural terrain classification using three-dimensional ladar data for ground robot mobility," *Journal of Field Robotics*, vol. 23, no. 10, pp. 839 – 861, November 2006.

[6] B. Taskar, "Learning structured prediction models: a large margin approach," Ph.D. dissertation, Stanford, CA, USA, 2005.

[7] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[8] S. Thrun, M. Montemerlo, and A. Aron, "Probabilistic terrain analysis for high-speed desert driving," in *Proceedings of Robotics: Science and Systems*, Philadelphia, USA, August 2006.

[9] L. G. Shapiro and G. Stockman, *Computer Vision*. Upper Saddle River, NJ: Prentice Hall, 2001.

[10] A. Frome, D. Huber, R. Kolluri, T. Bulow, and J. Malik, "Recognizing objects in range data using regional point descriptors," in *Proceedings of the European Conference on Computer Vision (ECCV)*, May 2004.

[11] A. Johnson and M. Hebert, "Using spin images for efficient object recognition in cluttered 3d scenes," vol. 21, no. 1, pp. 433 – 449, May 1999.

[12] B. Schölkopf, A. Smola, R. C. Williamson, and P. L. Bartlett, "New support vector algorithms," *Neural Computation*, vol. 12, no. 5, pp. 1207–1245, 2000.

[13] R. Gilad-Bachrach, A. Navot, and N. Tishb, "Margin Based Feature Selection - Theory and Algorithms ," in *Proceedings of the 21 st International Conference on Machine Learning, Banff, Canada, 2004*, 2004.

[14] Y. Bar-Shalom, T. Kirubarajan, and X.-R. Li, *Estimation with Applications to Tracking and Navigation*. New York, NY, USA: John Wiley & Sons, Inc., 2002.

[15] S. J. Julier and J. K. Uhlmann, "New extension of the Kalman filter to nonlinear systems," in *Proc. SPIE Vol. 3068, p. 182-193, Signal Processing, Sensor Fusion, and Target Recognition VI*, I. Kadar, Ed., vol. 3068, July 1997, pp. 182–193.

[16] F. von Hundelshausen, M. Himmelsbach, F. Hecker, A. Mueller, and H.-J. Wuensche, "Driving with tentacles: Integral structures for sensing and motion," *J. Field Robot.*, vol. 25, no. 9, pp. 640–673, 2008.

[17] D. Cole, A. Harrison, and P. Newman, "Using Naturally Salient Regions for SLAM with 3D Laser Data," in *Proc. International Conference on Robotics and Automation, ICRA, 2005*, 2005.

[18] R. Unnikrishnan and M. Hebert, "Multi-Scale Interest Regions from Unorganized Point Clouds," in *Workshop on Search in 3D (S3D), IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2008.