In Proceedings of 15th RoboCup International Symposium, Istanbul, 2011.
In: RoboCup 2011: Robot Soccer World Cup XV, LNCS 7416, pp. 306-317, Springer, 2012.

Real-Time Plane Segmentation using RGB-D Cameras*

Dirk Holz[1], Stefan Holzer[2], Radu Bogdan Rusu[3], and Sven Behnke[1]

[1] Autonomous Intelligent Systems Group, University of Bonn, Germany
holz@ais.uni-bonn.de, behnke@cs.uni-bonn.de
[2] Department of Computer Science, Technical University of Munich (TUM), Germany
holzers@in.tum.de
[3] Willow Garage, Inc., Menlo Park, CA, USA
rusu@willowgarage.com

Abstract. Real-time 3D perception of the surrounding environment is a crucial precondition for the reliable and safe application of mobile service robots in domestic environments. Using a RGB-D camera, we present a system for acquiring and processing 3D (semantic) information at frame rates of up to 30Hz that allows a mobile robot to reliably detect obstacles and segment graspable objects and supporting surfaces as well as the overall scene geometry. Using integral images, we compute local surface normals. The points are then clustered, segmented, and classified in both normal space and spherical coordinates. The system is tested in different setups in a real household environment.
The results show that the system is capable of reliably detecting obstacles at high frame rates, even in case of obstacles that move fast or do not considerably stick out of the ground. The segmentation of all planes in the 3D data even allows for correcting characteristic measurement errors and for reconstructing the original scene geometry in far ranges.

1 Introduction

Perceiving the geometry of environmental structures surrounding the robot is a crucial prerequisite for the autonomous operation of service robots in human living environments. These environments tend to be cluttered and highly dynamic. The first property requires for three-dimensional information about the environmental structures and objects contained therein, whereas the latter necessitates real-time processing of the acquired spatial information.

Recent RGB-D cameras, such as the Microsoft Kinect camera, acquire both visual information (RGB) like regular camera systems, as well as depth information (D) at high frame rates. In terms of measurement accuracy in low ranges (up to a few meters), the acquired depth information does not rank behind the accuracy achieved with 3D laser range scanners. With respect to the frame rate, RGB-D cameras clearly outperform commonly used 3D laser range scanners, e.g.,

* This research has been partially funded by the FP7 ICT-2007.2.1 project ECHORD (grant agreement 231143) experiment ActReMa.

30Hz vs. 1Hz. However, for applying and making use of these cameras in typical mobile manipulation problems like detecting objects and collision avoidance, the acquired RGB-D camera data needs to be processed in real-time (possibly with limited computing power). We present a system of algorithms for processing 3D point clouds in real-time which explicitly makes use of the organized structure of RGB-D camera data. It allows for

1. reliably detecting obstacles,
2. detecting graspable objects as well as the planes supporting them, and
3. segmenting and classifying all planes in the acquired 3D data.

An important characteristic of the proposed system is that the all of the above outcomes can be obtained at frame rates of up to 30Hz. In addition to the organized data structure, we exploit a specific characteristic of man-made environments, namely being primarily composed of connected planes like walls, ground floors, ceilings, tables etc. In fact, the work presented in this paper is solely based on a fast segmentation of all planes in 3D point clouds.

The remainder of this paper is organized as follows: After giving an overview on related work in Section 2, we discuss methods for computing local surface normals and present a fast variant using integral images in Section 3. Clustering these normals and segmenting all planes in acquired point clouds is described in Section 4. Detecting graspable objects and obstacles based on this information is discussed in Section 5. Section 6 summarizes experimental results.

2 Related work

Autonomous robot operation in complex real-world environments requires for powerful perception capabilities. 3D semantic perception has seen considerable progress recently. Here, we focus on two aspects – extracting semantic information from 3D data and using 3D data for obstacle detection and collision avoidance.

Safety-critical tasks like collision avoidance necessitate a fast perception of obstacles and processing acquired sensor data in real-time. A common way of using 3D data for collision avoidance in the navigation domain is to extract relevant information from the 3D input and to project it down to *virtual* 2D laser range scans for which navigation and collision avoidance are well studied topics. Wulf *et al.* use these virtual scans for both collision avoidance and localization in ceiling structures [12]. Holz *et al.* distinguish two types of virtual scans, virtual structure and obstacle maps. The first type models environmental structures such as walls in a virtual 2D laser range scan, the latter information about closest obstacles [3]. Yuan *et al.* follow this approach in [13] to fuse sensor information from a Time-of-Flight (ToF) camera with that of a 2D laser range scanner to compensate for the smaller field of view of ToF cameras. Droeschel *et al.* only use a ToF camera for obstacle detection, but mount this camera on a pan-tilt-unit and use an active gaze control to keep relevant regions in sight [1]. Measured points sticking out of the ground are considered as obstacles. This information is

then fused with other 2D laser range scanners on the robot, again, by projecting them into a virtual 2D laser scan. Problems arise in regions where the ToF data is highly affected by noise and in case of motion blur. In both cases, floor points may be considered as obstacles. Furthermore, obstacles whose size lies below the measurement accuracy and the used thresholds to compensate for noisy data cannot be detected. In contrast, we consider as obstacles both points sticking out of segmented planes as well as points with different local surface orientations. This allows for detecting even small objects reliably as obstacles.

Nüchter *et al.* extract environmental structures such as walls, ceilings and drivable surfaces from 3D laser range scans and use trained classifiers to detect objects like, for instance, humans and other robots [8]. They examine range differences in consecutive laser scan points to get an approximate estimate if a point is lying on a horizontal or on a vertical surface. Triebel *et al.* use Conditional Random Fields to segment and discover repetitive objects in 3D laser range scans [11]. Rusu *et al.* extract hybrid representations of objects consisting of detected shapes, as well as surface reconstructions where no shapes have been detected [9]. Endres *et al.* extract objects and point clusters from the background structure in range scans (assuming objects being spatially disconnected) [2]. Using latent Dirichlet allocation, they can derive classes of objects from these clusters that are similar in shape. Lai and Fox use data from Google's 3D Warehouse to classify clusters of points in 3D laser range scans [5]. They detect walls, trees and cars in street scenes. Pathak *et al.* decompose acquired 3D data into plane segments and use this information for registering point clouds. Steder *et al.* compute range images from point clouds, extract borders and key-points and use 3D feature descriptors to find and match repetitive structures [10]. The above approaches show good results when processing accurate 3D laser range data, but tend to have high runtime requirements. Here we focus on less complex but fast methods to obtain an initial segmentation of the environment in real-time that can be used in a variety of applications.

3 Fast computation of local surface normals

Local geometric features such as surface normal or curvature at a point form a fundamental basis for extracting semantic information from 3D sensor data. A common way for determining the normal to a point $\boldsymbol{p}_i$ on a surface is to approximate the problem by fitting a plane to the point's local neighborhood $\mathcal{P}_i$. This neighborhood is formed either by the k nearest neighbors of $\boldsymbol{p}_i$ or by all points within a radius r from $\boldsymbol{p}_i$. Given the $\mathcal{P}_i$, the local surface normal $\boldsymbol{n}_i$ can be estimated by analyzing the eigenvectors of the covariance matrix $C_i \in \mathbb{R}^{3\times3}$ of $\mathcal{P}_i$. The eigenvector $\boldsymbol{v}_{i,0}$ corresponding to the smallest eigenvalue $\lambda_{i,0}$ can be used as an estimate of $\boldsymbol{n}_i$. The ratio between $\lambda_{i,0}$ and the sum of eigenvalues provides an estimate of the local curvature.

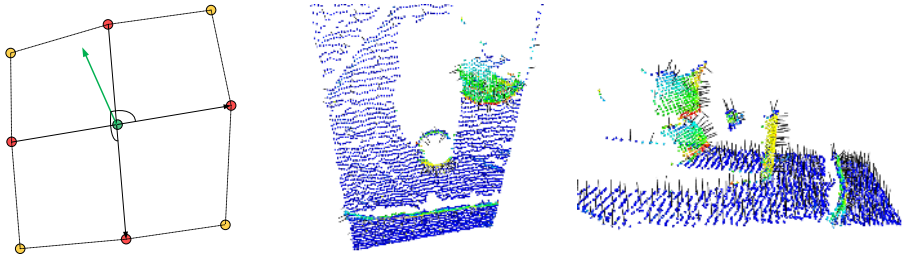
Both k and r highly influence how well the estimated normal represents the local surface at $\boldsymbol{p}_i$. Chosen too large, environmental structures are considerably smoothened so that local extrema such as corners completely vanish. If the

neighborhood is too small, the estimated normals are highly affected by the depth measurement noise. A common way to compensate these effects that is also used in [9] is to compute the distances of all points in $\mathcal{P}_i$ to the local plane through $\boldsymbol{p}_i$. These distances are then used in a second run to weight the points in $\mathcal{P}_i$ in the covariance computation. By this means, corners and edges are less smoothened and the estimated normals better approximate the local surface structure. However, with or without this second run, estimating the point's local neighborhood is computationally expensive, even when using approximate search in kD-trees which is $O(n \log n)$ for n randomly distributed data points (plus the construction of the tree). Another possibility to compensate for the aforementioned effects is to compute the normals in different neighborhood ranges or different scales of the input data and to select the most likely surface normal for each point.

Less accurate, but considerably faster is to consider pixel neighborhoods instead of spatial neighborhoods [4]. That is, the organized structure of the point cloud as acquired by Time-of-Flight or RGB-D cameras is used instead of searching through the 3D space spanned by the points in the cloud. Compared to a fixed radius r or a fixed number of neighbors k, using a fixed pixel neighborhood has the advantage of having smaller neighborhoods in close range (causing more accurate normals) and larger neighborhoods smoothing the data in far ranges that is more affected by noise and other error sources (see, e.g., [6] for an overview on Time-of-Flight camera error sources).

By using a fixed pixel neighborhood and, in addition, neglecting pre-computed neighbors outside of some maximum range r as in [4], one can avoid the computationally expensive neighbor search, but still needs to compute and analyze the local covariance matrix. Here, we use an approach that directly computes the normal vector over the neighboring pixels in x and y image space.

The basic principle of our approach is to compute two vectors which are tangential to the local surface at the point $\boldsymbol{p}_i$. From these two tangential vectors we can easily compute the normal using the cross product. The simplest approach for computing the normals is to compute them between the left and right neighboring pixel and between the upper and lower neighboring pixel, as illustrated in Figure 1.a. However, since we expect noisy data and regions in which no depth information is available (a special characteristic of the used cameras), the resulting normals would also be highly affected. For this reason, we apply a smoothing on the tangential vectors by computing the average vectors within a certain neighborhood. To perform this smoothing efficiently we use integral images. We first create two maps of tangential vectors, one for the x- and one for the y-direction (again in image space). The vectors for these maps are computed between corresponding 3D points in the point cloud. That is, each element of theses vectors is a 3D vector. For each of the channels (Cartesian x, y, and z) of each of the maps we then compute an integral image, which leads to a total number of six integral images. Using these integral images, we can compute the average tangential vectors with only $2 \times 4 \times 3$ memory accesses, independent of the size of the smoothing area. The overall runtime complexity is linear in the number of points for which normals are computed (cf. Figure 6.a).



(a) Basic principle (b) Example (left: top view, right: side view)

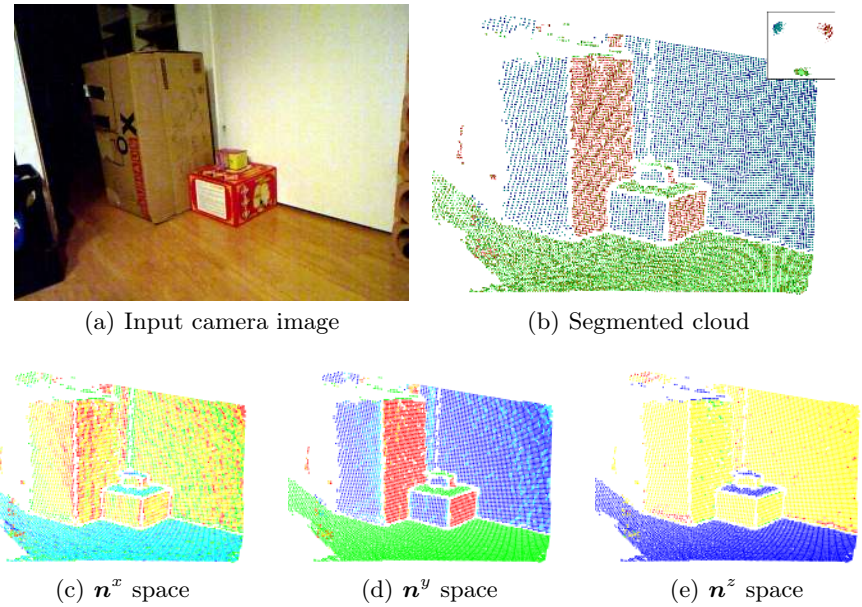
Fig. 1. Principle of fast normal computation using integral images (a). Two vectors tangential to the surface at the desired position are computed using the red points. The local surface normal is computed by applying the cross product to them. A typical result of an acquired point cloud with surface normals is shown in (b).

The computation of normals is conducted in the local image coordinate frame ($\hat{Z}$-axis pointing forwards in measurement direction). For further processing, we transform both Cartesian coordinates of the points as well as the local surface normals into the base coordinate frame of the robot (right-handed coordinate frame with the $\hat{X}$-axis pointing in measurement and driving direction, the $\hat{Z}$-axis point upwards representing the height of points). In case this transformation is not known, we only apply the corresponding reflection matrix and a translation by 2cm along the $\hat{Y}$-axis that accounts for the difference in position between the regular camera and the infrared camera that senses the emitted pattern for depth reconstruction. It should be noted that this transformation (or the knowledge of the camera's position and orientation in space) is not necessary for the fast plane segmentation in Section 4, but only for task-specific applications like the extraction of horizontal surfaces.

In addition to the fast normal estimation, we compute spherical coordinates (r, ϕ, θ) of the local surface normals that ease the classification of measured points and the processing steps presented in the following. We define ϕ as the angle between the local surface normal (projected onto the $\hat{X}\hat{Y}$-plane) and the $\hat{X}$-axis, r the distance to the origin in normal space, and θ the angle between the normal and the $\hat{X}\hat{Y}$-plane. (r, ϕ) is of special interest in obstacle detection, as it represents direction and distance of an obstacle to the robot (in the $\hat{X}\hat{Y}$-plane). For plane segmentation, r is abused in our implementation to hold the plane's distance from the origin (in Cartesian space).

4 Fast plane segmentation

Man-made environments tend to be primarily composed of planes. Detected and segmented planes already adequately model the surface of most environmental structures. We segment local surface normals in two steps: 1) we cluster (and merge) the points in normal space $(\boldsymbol{n}^x, \boldsymbol{n}^y, \boldsymbol{n}^z)^T$ to obtain clusters of plane



(a) Input camera image (b) Segmented cloud

(c) $\boldsymbol{n}^x$ space (d) $\boldsymbol{n}^y$ space (e) $\boldsymbol{n}^z$ space

Fig. 2. Typical result of the first segmentation step: points with similar surface normal orientation in the input data (a) are merged into clusters (b, shown in both Cartesian and normal space). The components of the normals ($\boldsymbol{n}^x$, $\boldsymbol{n}^y$, and $\boldsymbol{n}^z$) are visualized in (c-e) using a color coding from -1 (red) to 1 (blue). This simple clustering already allows for a fast segmentation of planes with similar surface normal orientation, e.g., extracting all horizontal planes.

candidates and 2) cluster (and merge) planes of similar local surface normal orientation in distance space (distance between plane and origin).

4.1 Initial segmentation in normal space

For the initial clustering step in which we want to find clusters of points with similar local surface normal orientations, we construct a voxel grid either in normal space or using the spherical coordinates. Using the spherical coordinates allows for clustering in the two-dimensional (ϕ, θ)-space, but requires a larger neighborhood in the subsequent processing step in which we merge clusters. Both results and processing time do not differ.

For clustering in normal space, we compute a three-dimensional voxel grid and map local surface normals to the corresponding grid cell w.r.t. the cell's size. Points for which the surface normals fall into the same cell, form the initial cluster and potential set of planes with the same normal orientation. Either all non-empty cells or only those with a minimum number of points are considered as initial clusters.

In order to compensate for the involved discretization effects, we examine the cell's neighbors in the three-dimensional grid structure. If the average surface

normal orientation in two neighboring grid cells falls below the cluster size (and the desired accuracy), the corresponding clusters are merged. For being able to merge multiple clusters, we keep track of the conducted merges. In case cluster a should be merged with cluster b that was already merged with cluster c, we check if we can merge a and c, or if $a + b$ is a better merge than $a + c$. Although this procedure is less adaptive (and complex) as sophisticated clustering algorithms like *k-Means*, *mean-shift*-clustering or, e.g., ISODATA [7], this simple approach allows for reliably detecting larger planes in 3D point clouds at high frame rates. In all modes and resolutions of the camera, plane segmentation is only a matter of milliseconds. In contrast to region growing algorithms, we find a singe cluster for planes that are not geometrically connected, e.g., parts of the same wall.

An example segmentation is shown in Figure 2. Planes with similar (or equal) local surface normal orientations are contained in the same cluster and visualized with the same color.

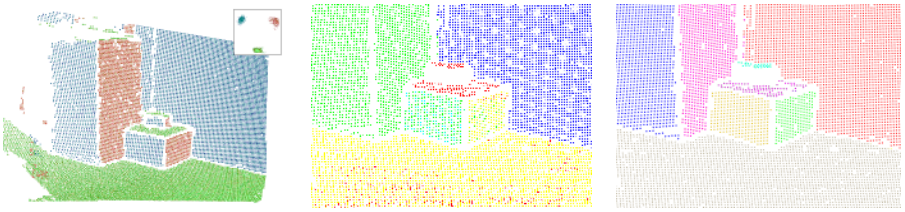
4.2 Segmentation refinement in distance space

Up to now the found clusters do not represent single planes but sets of planes with similar or equal surface normal orientation. For some applications like extracting all horizontal surfaces, this information can directly be used. For other applications, we split these normal clusters into plane clusters such that each cluster resembles a single plane in the environment.

Under the assumption that all points in a cluster are lying on the same plane, we use the corresponding averaged and normalized surface normal to compute the distance from the origin to the plane through the point under consideration. Naturally these distances differ for points on different parallel planes and we can split clusters in distance space. For compensating the fact that measurements farther away from the sensor are stronger affected by the different error and noise sources, we compute a logarithmic histogram. Again, points whose distances fall into the same bin form initial clusters. These clusters are then refined by examining the neighboring bins just like in the refinement of the normal segmentation. An example of the resulting plane clusters is shown in Figure 3.

5 Applications

The planes segmented at frame rates of up to 30Hz are useful for a variety of applications. Here, we use the plane segments as well as the computed surface normals and spherical coordinates for detecting obstacles and graspable objects in table top scenes and on the ground floor. In addition, we can compensate for camera-specific noise and error sources by projecting all points onto the planes they belong to.



(a) 3 Normal clusters (b) Distance space (c) 8 Plane clusters

Fig. 3. Typical result of the second segmentation step: Clusters with similar surface normal orientations (a) are clustered in distance space (b). Clusters with similar normal orientations but varying distances (of the respective planes to the origin) are split. For compensating discretization effects, neighboring clusters are again merged to form the segmented planes (c). Color coding in (a+c) is random per cluster, and in distance space from 0.4m (red) to 1.3m (blue) in (b).

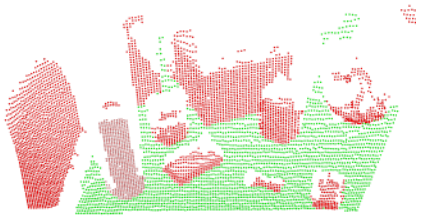
5.1 Obstacle and object detection

For detecting obstacles and graspable objects, we first extract all horizontal plane segments, i.e., those with $\boldsymbol{n}^x \approx 1$ (and $\theta \approx +\frac{pi}{2}$ respectively). That is, we exploit the fact that both the robot as well as objects in its environments are standing on (or *supported* by horizontal surfaces). Hence, we need a rough estimate of the camera orientation in order to determine which planes are horizontal. Furthermore, depending on the robot's task, e.g., navigation or manipulation of objects on a table, we limit the height in which we search for horizontal planes.

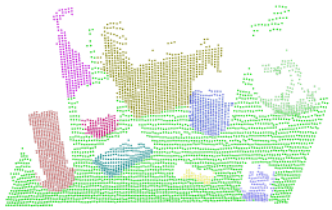
For navigation purposes, only the ground floor plane ($\boldsymbol{n}^z \approx 1,\ z \approx 0$) is considered safe. All other points and planes including other horizontal surfaces such as tables are considered as obstacles. For object detection, we limit the search space by the height range in which the robot can manipulate. Since our robots are equipped with a trunk that can be lifted and twisted [reference removed], we use a range of $0m - 1.2m$.

We follow a similar approach as in [9] and [4] for detecting objects. The already found plane model (consisting of the averaged normals and plane-origin distances in a plane cluster) is optimized using a RANSAC approach that also sorts out residual outliers. We then project all cluster points onto the plane and compute the convex hull. These steps are repeated for all horizontal planes that have been found in the given height range. For all points from non-horizontal plane clusters, we then check if they lie above a supporting plane (within a range of e.g. 30cm) and within the corresponding convex hull (again with a tolerance of a few centimeters). Points meeting both requirements are then clustered to obtain object candidates. For each of the candidates we compute the centroid and the oriented bounding box in order to distinguish graspable from non-graspable objects. Here we simply assume that the minimum side length of graspable objects needs to lie between 1 and 10cm. Furthermore, we neglect clusters where the number of contained points falls below a threshold (e.g. 50 points). Figure

(a) Example table scene



(b) Detected obstacles



(c) Detected objects

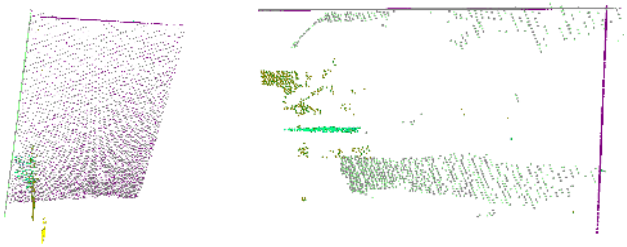
Fig. 4. Typical result of detecting obstacles and objects in a table top setting (a). Even obstacles (b, red) that, in 3D, do not stick out of the supporting surface (b, green) like the red lighter are perceived. Detected objects (c) are randomly colored. For being able to grasp an object, the respective cluster is not considered as an obstacle (in this example the Pringles box).

4 shows a typical result of detecting graspable objects and obstacles in a table top setting.

5.2 Correcting local surfaces at detected planes

RGB-D cameras suffer from different noise and error sources, especially discretization effects in depth measurements and the fact that the cameras are calibrated for a certain range. Both effects cause considerable measurement errors in far ranges (e.g. >3.5m). Especially for modeling the geometry of environmental structures where planes are of special interest, these measurements hinder from finding accurate surface models. However, with the extracted plane clusters, we can project the contained points onto the corresponding plane to get, at least, an approximate estimate of the true surface geometry. Figure 5 shows a typical result of this naïve measurement correction which considerably increases the quality of acquired geometric information. In fact, the angle between the two walls in Figure 5.b only deviates by 2° from ground truth. However, it is a matter of future work to take this information into account in a precise and adaptive sensor model.

(a) Input cloud (left: top view, right: side view)



(b) Output cloud (left: top view, right: side view)

Fig. 5. Typical result of correcting local surface geometry. Shown are the segmented input cloud (a) and the corrected cloud (b). The views are rotated to be aligned with the plane tangents of a wall and the ceiling. The distance to the wall (magenta) is approximately 4m.

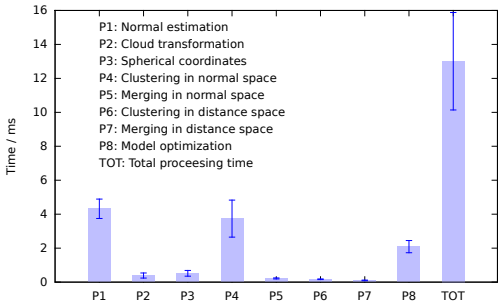
6 Experiments

Both the detection of graspable objects and obstacles as well as the naïve correction using plane segments highly depend on the quality of the estimated surface normals. In order to evaluate the accuracy of the estimated normals, we conducted a first sequence of experiments comparing the estimated surface normal at each point with the one computed over the real neighbors using the two-run Principal Component Analysis as described above. For the neighbor search a radius r has been chosen that linearly depends on the measured range to the point under consideration, i.e., a smaller radius for the more accurate close range measurements and a larger radius for measurements being farther apart from the sensor. This radius function has been manually adapted for each of the point clouds used in the experiments in order to guarantee correct (and ground truth-like) normals.

The presented results have been measured over 40 points clouds taken in 4 different scenes in a real house-hold environment: 1) a table top scene with only one object, 2) a cluttered table top scene with $>$50 objects, 3) a room with a cluttered table top and distant walls, 4) a longer corridor with several cabinets where measurement of up to 6m have been taken. In average, a deviation of roughly 10° has been measured (see Figure 6(a)). This is primarily caused by the fact that

Resolution	Processing time	Avg. Deviation	Considerable deviations
640 × 480 (VGA)	61.84 ± 7.73 ms	11.75 ± 3.02 deg	roughly 2%
320 × 240 (QVGA)	15.08 ± 2.19 ms	12.39 ± 3.81 deg	roughly 1.2%
160 × 120 (QQVGA)	4.32 ± 0.57 ms	9.32 ± 2.44 deg	roughly 1%

(a) Processing time and accuracy for normal estimation



(b) Processing times (QQVGA)

Resolution	VGA	QVGA	QQVGA
obstacles	100%	100%	100%
!*obstacles*	0.1%	0.15%	0.19%
objects	92%	93%	95%
!*objects*	0%	0%	0%
Rate	≈7Hz	≈27Hz	**≈30Hz**

(c) Detection rates

Fig. 6. Run-times for all processing steps (a+b) and reliability of object/obstacle detection (c). 100% of obstacles are perceived (*obstacles*), and only !*obstacles* % of the measurements have been incorrectly classified as obstacles. Roughly 93% (*objects*) of the objects have been correctly detected, and only !*objects* % points have been segmented as belonging to a non-existent object. "Rate" is the frequency with which the results are provided to other components in the robot control architecture.

the current implementation does not specifically handle edges and corners as is done with the second PCA run on the weighted covariance. Furthermore, using nearest neighbor search better compensates for missing measurements in regions where no depth information is available. However, especially in close range (e.g. up to 2m), the estimated normals are quite accurate and do not deviate from the *true* local surface normals. Only one to two percent of the estimated normals considerably deviated from the true normals (deviations larger than 25°).

In all experiments, processing times have been measured on a Core i7 machine and over several minutes, i.e., several thousand point clouds. No parallel computation has been carried out and all algorithms were run sequentially within a single thread on a single core. That is, processing times should not considerably deviate on (newer) notebook computers. Figure 6.b summarizes all results. Obstacles were detected always (100%) and 93% of the objects have been correctly segmented. The object segmentation gets inaccurate if objects are 1) very small (dimensions falling below the aforementioned thresholds), or 2) being more than 3.5m away from the sensor (where depth measurements are highly inaccurate).

7 Conclusion

We have presented an approach to real-time 3D point cloud processing that segments planes in the space of local surface normals. The detected planes have

been used for detecting graspable objects and obstacles as well as for correcting the measured 3D information. With frame rates of up to 30Hz we can reliably detect obstacles in the robot's vicinity as well as objects for manipulation tasks. However, it remains a matter of future work to find reliable and fast approaches to more complex tasks like autonomous registration of point clouds or recognizing detected objects that make use of the acquired surface information.
Data sets and videos are available at: `http://purl.org/holz/segmentation`.

References

1. D. Droeschel, D. Holz, J. Stückler, and S. Behnke. Using Time-of-Flight Cameras with Active Gaze Control for 3D Collision Avoidance. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2010.
2. F. Endres, C. Plagemann, C. Stachniss, and W. Burgard. Unsupervised discovery of object classes from range data using latent dirichlet allocation. In *Proc. of Robotics: Science and Systems*, 2009.
3. D. Holz, C. Lörken, and H. Surmann. Continuous 3D Sensing for Navigation and SLAM in Cluttered and Dynamic Environments. In *Proc. of the International Conference on Information Fusion (FUSION)*, 2008.
4. D. Holz, R. Schnabel, D. Droeschel, J. Stückler, and S. Behnke. Towards Semantic Scene Analysis with Time-of-Flight Cameras. In *Proc. of the 14th RoboCup International Symposium*, 2010.
5. K. Lai and D. Fox. 3D laser scan classification using web data and domain adaptation. In *Proc. of Robotics: Science and Systems*, 2009.
6. S. May, D. Droeschel, D. Holz, S. Fuchs, E. Malis, A. Nüchter, and J. Hertzberg. Three-dimensional mapping with time-of-flight cameras. *Journal of Field Robotics, Special Issue on Three-Dimensional Mapping, Part 2*, 26(11-12):934–965, 2009.
7. N. Memarsadeghi, D. M. Mount, N. S. Netanyahu, and J. Le Moigne. A fast implementation of the isodata clustering algorithm. *International Journal of Computational Geometry and Applications*, 17:71–103, 2007.
8. A. Nüchter and J. Hertzberg. Towards semantic maps for mobile robots. *Robotics and Autonomous Systems*, 56(11):915–926, 2008.
9. R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz. Close-range Scene Segmentation and Reconstruction of 3D Point Cloud Maps for Mobile Manipulation in Human Environments. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2009.
10. B. Steder, R. B. Rusu, K. Konolige, and W. Burgard. NARF: 3D range image features for object recognition. In *Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010.
11. R. Triebel, J. Shin, and R. Siegwart. Segmentation and unsupervised part-based discovery of repetitive objects. In *Proc. of Robotics: Science and Systems*, 2010.
12. O. Wulf, K. O. Arras, H. I. Christensen, and B. Wagner. 2D Mapping of Cluttered Indoor Environments by Means of 3D Perception. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2004.
13. F. Yuan, A.s Swadzba, R. Philippsen, O. Engin, M. Hanheide, and S. Wachsmuth. Laser-based navigation enhanced with 3D time-of-flight data. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2008.