Efficient Organized Point Cloud Segmentation with Connected Components

Alexander J. B. Trevor, Suat Gedikli, Radu B. Rusu, Henrik I. Christensen

***Abstract*— Segmentation is an important step in many perception tasks, such as object detection and recognition. We present an approach to organized point cloud segmentation and its application to plane segmentation, and euclidean clustering for tabletop object detection. The proposed approach is efficient and enables real-time plane segmentation for VGA resolution RGB-D data. Timing results are provided for indoor datasets, and applications to tabletop object segmentation and mapping with planar landmarks are discussed.**

I. Introduction

Segmentation is an important step in many perception tasks, such as some approaches to object detection and recognition. Segmentation of images has been widely studied in the computer vision community, and point cloud segmentation has also been of interest. Planar segmentation of point cloud data has been of particular interest, as this can be helpful for a variety of tasks. Detection of tabletop objects by extracting contiguous clusters of points above planar surfaces such as tables was proposed by Rusu *et. al* [9]. Segmented planar surfaces have also been used as landmarks for feature-based SLAM [12] and semantic mapping applications [13].

In this paper, we describe an efficient method for segmenting organized point cloud data. The image-like structure of organized point clouds enables us to apply approaches from computer vision, including graph-based or connected-component segmentation approaches. However, in addition to RGB data as would be available in an image, we also make use of the 3D coordinates of each pixel / point, as well as surface normal information.

The paper is structured as follows: Section II describes selected related works from both image segmentation and point cloud segmentation. Section III describes our approach to segmentation, including a detailed description of our algorithm and applications to several tasks. Section IV describes and open source implementation of our approach, while Section V includes a discussion as well as run time results.

II. Related Work

Segmentation of images, range images, and point clouds have all been widely studied in the literature. Several of the most closely related works will be highlighted here.

Felzenszwalb and Huttenlocher [2] proposed a graph-based approach to image segmentation that shares some similarities with our approach. A graph structure is imposed on the image, such as a 4-connected grid, and various predicates are used to compute edge weights. Different graph structures and predicates can be used to segment in different ways. Our approach defines predicates in a similar way to operate on our image and point cloud data, but uses a fixed 4-connected graph structure.

A. Trevor and H. I. Christensen are affiliated with the Georgia Institute of Technology's center for Robotics & Intelligent Machines. {atrevor, hic}@cc.gatech.edu. S. Gedikli and R. B. Rusu are affiliated with Open Perception.

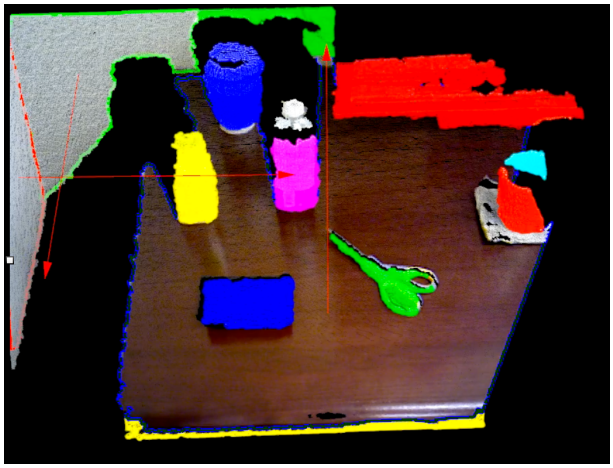

Fig. 1. An example of a segmented tabletop scene. Planar surfaces are outlined in color, with large red arrows indicating the segmented plane's normal. Euclidean clusters are indicated by colored points.

Several approaches to planar segmentation of point cloud data have also been proposed. Many RANSAC-based approaches such as in [8] can be used to accurately segment planar surfaces, but these approaches were designed for unorganized point clouds as are produced by 3D laser scans, and so are much slower than approaches that can exploit organized data. Other approaches to unorganized point cloud segmentation have been surveyed in [14]. Some RANSAC-based approaches have been extended to exploit organized structure, such as the approach of Biswas and Veloso [1], which enables real-time performance.

Organized point cloud and range image segmentation has also been investigated. Poppinga *et. al* [7] proposed a region-growing based approach that selects a random point, and grows a region around it to the nearest neighbors that are nearby in plane space, incrementally updating the centroid and covariance matrix as points are added. Holz *et. al* extend this approach by pre-computing surface normals for the cloud, and incrementally update the plane's normal equation, which further reduces the computational cost [4] [3].

In contrast to these methods, our approach does not utilize seed points for region growing, but instead process each point sequentially. Additionally, we do not incrementally compute

any statistics per region, instead delaying plane fitting until the image has been fully segmented, so that processing need only be performed for segments with a sufficient number of inlying points.

Hulik *et. al* evaluate several approaches to depth image segmentation, including a connected-component-like approach that operates on edge images [6]. This was reported to be quite efficient, but less accurate than some alternative methods such as RANSAC, for the datasets used in their testing.

III. Approach

A. Connected Component Algorithm

We propose a connected component based approach to segmentation which operates on organized point cloud data. An organized point cloud is a point cloud that has an image-like grid structure. We will denote an organized point cloud as $P(x, y)$, indicating the x and y coordinates of the point in the image-like structure. As in an image, given a point $P(x, y)$, neighboring points such as $P(x-1, y)$ and $P(x, y-1)$ are easily accessible in constant time thanks to the cloud's ordering. This organization is exploited by our segmentation approach, enabling us to skip the costly neighborhood step common to many segmentation approaches.

Different segmentation tasks may require different point representations, which may include RGB data as would be present in a color image, points in euclidean space as are commonly used in point clouds, as well as information like surface normals. Some types of information may not be available for each point in the organized structure. For example, RGB-D sensors often return invalid depth measurements, so no euclidean point is available. In these cases, a value such as NaN is used to signify that there is invalid data while maintaining the organized structure. For the purposes of segmentation, if required information such as a depth value or surface normal is not available for a point, the point will not be added to any segment.

The algorithm works by partitioning an organized point cloud P into a set of segments S. This is done by creating an integer label L for each point in the point cloud. This can be thought of as a label image. The label for a point at $P(x, y)$ will be denoted as $L(x, y)$. Points that have missing or invalid data can be excluded by using an invalid label such as NaN or maxint. Points that are members of the same segment will be assigned the same label. That is, if $P(x_1, y_1) \in S_i$ and $P(x_2, y_2) \in S_i$, then $L(x_1, y_1) = L(x_2, y_2)$. Points are compared using a comparison function or predicate. The exact comparison function will vary for different segmentation tasks, but it is of the form:

$$C(P(x_1, y_1), P(x_2, y_2)) = \begin{cases} true\ if\ similar \\ false\ otherwise \end{cases}$$

If $C(P(x_1, y_1), P(x_2, y_2)) = true$, then $L(x_2, y_2) = L(x_1, y_1)$, else $L(x_2, y_2) \neq L(x_1, y_1)$. In the latter case, a new label is created by incrementing the largest assigned label by one. The exact comparison function C will vary based on the segmentation task, and several such functions will be described below.

The connected component labeling algorithm we use is similar to the two-pass binary image labeling algorithm described in [10], but has been modified to label point cloud data with continuous values based on some predicate, as in [2]. The algorithm begins by assigning the first point in the cloud with valid data with label 0. The first row and column of the cloud are then compared with the specified comparator C, to assign labels. The remainder of the points are treated by examining their neighboring points $P(x-1, y)$ and $P(x, y-1)$, as in a 4-connected grid. If both neighbors (above and to the left) have different labels, these labels must be merged with that of the current pixel, as these should be part of the same segment. An example of such a situation is shown in Figure 2, where a segment on the current row and previous row must be merged. We use the union-find algorithm to do this efficiently, as in [10]. Once the label image has been created, a second pass is performed to merge labels, assigning the lowest applicable label to the region, and producing the final connected-component label image $L(x, y)$.

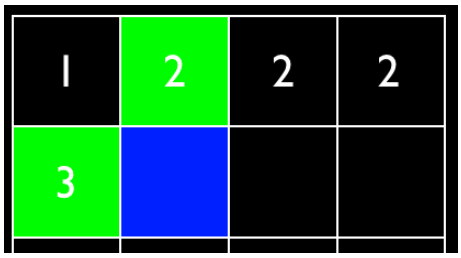

Fig. 2. An example where two labels must be merged, if the pixel highlighted in blue matches both the neighbor above and to the left, shown in green. Merging is performed in a second pass using the union-find algorithm.

Many tasks can benefit from the boundary points of a segment. Given a segment, it is straightforward to compute the boundary by tracing the outer contour. Such an approach is also included in our implementation.

B. Planar Segmentation

Our planar segmentation approach segments the scene to detect large connected components corresponding to planar surfaces, including walls, the ground, tables, etc. We use the hessian-normal form to represent planes, which uses the well known equation $ax + by + cz + d = 0$. Our approach to planar segmentation begins by computing such a planar equation for each point in Euclidean space that has a valid surface normal.

Surface normals can be computed for the cloud using a variety of techniques. In this work, we use the integral image normal estimation method of Holzer *et. al* [5], which can compute normals in real-time for VGA RGB-D point clouds. After computing unit-length normals $\{n_x, n_y, n_z\}$ for each point, a point p can be represented as:

$$p = \{x, y, z, n_x, n_y, n_z\}$$

Additionally, given such a Euclidean point with its normal, we compute the perpendicular distance to this point with normal (the d variable of the plane equation), giving us a coordinate in plane space. This can be computed by the dot product:

$$n_d = \{x, y, z\} \cdot \{n_x, n_y, n_z\}$$

Augmenting our point representation with this information yields a point with a full plane equation:

$$p = \{x, y, z, n_x, n_y, n_z, n_d\}$$

Given this point representation, we then define distance metrics for both the normal direction and perpendicular distance components between two points. The distance in the range or d components of the plane equations is straightforward, as these are distances. The angular difference $dist_{normal}$ between the normal directions is given by the dot product.

$$dist_{normal}(p_1, p_2) = p_{1n} \cdot p_{2n}$$

$$dist_{range}(p_1, p_2) = |p_{1n_d} - p_{2n_d}|$$

We can then proceed with the connected component algorithm as described above, using a comparison function designed for planar segmentation. We first detect surfaces with smoothly changing surface normals, by requiring that surface normals of neighboring points are similar, within a threshold $thresh_{normal}$. Note that to avoid computing an inverse cosine for each point, we instead define the threshold $thresh_{normal}$ as the cosine of the desired angular threshold in radians.

$$C(p_1, p_2) = \begin{cases} true\ if((dist_{normal} < thresh_{normal}) \\ \quad \&\&\ (dist_{range} < thresh_{range})) \\ false\ otherwise \end{cases}$$

Using this comparison function with the above algorithm results in a set of labeled segments $L(x, y)$ corresponding to connected components in plane space. A visualization of such a label image is shown in Figure 8. Note that at this point, we have only examined local information, meaning that our segments may be only locally planar. The approach so far can be thought of as "smooth surface" segmentation rather than planar segmentation.

Next, we attempt a least squares plane fit for each segment with more than $min\_inliers$ points, resulting in a plane equation for each large segment. To ensure that the resulting segments are actually planar, the curvature is also computed, and a threshold $max\_curvature$ is used to filter out segments that are smooth but not planar.

C. Planar Refinement Algorithm

One shortcoming of the above approach is that it requires accurate surface normals, which are not always available. In particular, points / pixels near object boundaries and image boundaries tend to have noisy surface normals or no surface normals, which leads to segmented planar regions that end before the edge of the actual planar surface, as can be seen in Figure 3.

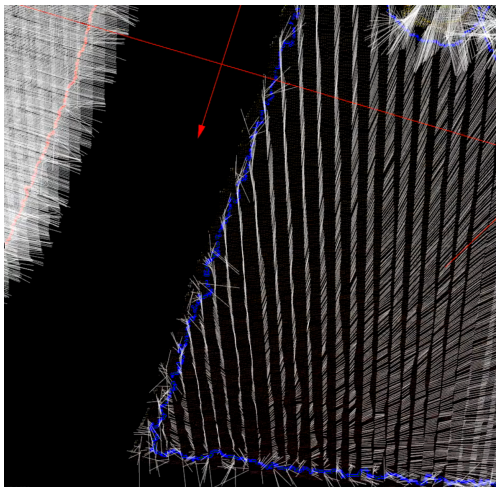

Fig. 3. An example scene with noisy surface normals near object boundaries.

This can be addressed by performing additional passes of the connected component algorithm with a new comparator that extends existing planar segments to include adjacent points (in a 4-connected sense) that have a point-to-plane distance under a given threshold to the adjacent segment's planar equation. Given a point with normal $p = \{x, y, z\}$ and a plane equation $eqn = \{n_x, n_y, n_z, n_d\}$, the point-to-to plane distance is given by:

$$dist_{ptp}(p, eqn) = \mid n_x * x + n_y * y + n_z * z + n_d \mid$$

The input to this comparison function requires the output labels L of our previous plane segmentation approach, as well as a set of labels $refine\_labels = \{l_1, ..., l_n\}$ which should be considered for refinement. Also required are the plane equations $eqns = \{eqn_1, ..., eqn_n\}$ corresponding to each segment label to be refined. Our planar refinement comparator works as follows:

$$C(p_1, p_2) = \begin{cases} false\ if\ (p_1 \notin refine\_labels \\ \quad \&\&\ p_2 \notin refine\_labels) \\ \quad ||\ dist_{ptp}(p_1, eqn(p_2)) > thresh_{ptp} \\ true\ otherwise \end{cases}$$

As this comparison only extends regions in one direction, two additional passes are required for refinement: a pass to extend planar regions "up and left", and a pass to extend "down and right". This is illustrated in Figure 5. While these additional passes do require computation, most points require very little process

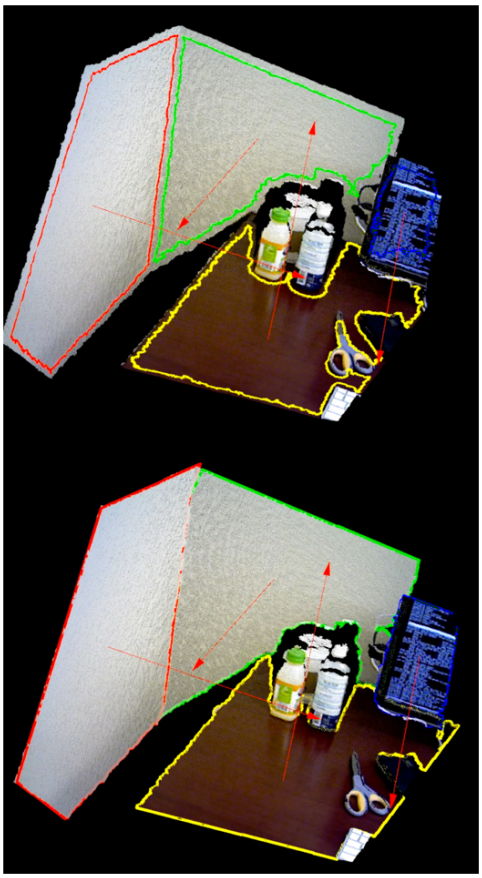

Fig. 4. Top: a scene segmented without using the planar refinement approach described in Section III-C. Bottom: a similar scene segmented with the planar refinement approach.

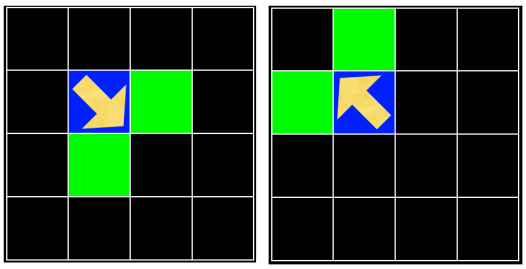

Fig. 5. Planar refinement requires two additional passes over the image, one that extends regions "down and right", and one that extends regions "up and left".

D. Euclidean Clustering

Similar to the algorithm described in [9], we take a greedy approach to euclidean clustering. The comparison function used is based on euclidean distance $d_{euclidean}$, given by the well known function.

To be useful for some tasks such as tabletop object detection, this approach also needs to be able to take an image mask. For computational reasons, we instead use a label image L from a previous step, as well as a set of labels $exclude\_labels$ to be included in the mask. For the purpose of tabletop object detection, we first segment planar surfaces, and use the planar regions as our mask – the corresponding labels are used as $exclude\_labels$. One could also construct a binary mask by making a label image with one label set as part of the mask. We can then define a comparison function as:

$$C(P_1, P_2) = \begin{cases} false\ if\ (L(P_1) \in exclude\_labels \\ \quad || \ L(P_2) \in exclude\_labels \\ \quad || \ d_{euclidean}(P_1, P_2) > d_{thresh}) \\ true\ otherwise \end{cases}$$

E. Planar Segmentation with Color

The segmentation approaches described above have only made use of depth information and surface normals. However, the approach can be extended to also consider color information in addition to other features. To segment planar regions of similar color, a new comparison function can be defined that is identical to the planar segmentation comparison function, but additionally requires that points have a distance in color space d_{color} below some threshold $thresh_{color}$. While perhaps not the best distance metric in color space, we used euclidean distance in RGB space to demonstrate this approach. An example scene is shown in Figure 6.

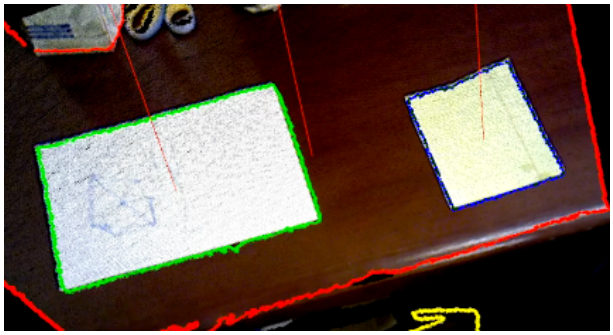

Fig. 6. An example scene segmented for planes of similar color. The sheets of paper on the desk are segmented as separate planar regions, as their color does not match the desk.

IV. Implementation

The above described approach is available in the Point Cloud Library (PCL) [8] as the Organized Connected Component Segmentation module. This class performs the segmentation algorithm described in Section III given some comparison function, such as the one described in Section III-B. To use a different comparison function, different comparator classes are used. A comparator contains a compare function that takes two points as parameters, and returns a boolean $true$ if the points should be members of the same segment, and $false$ otherwise. The class may contain additional information such as thresholds or a mask, as used in the clustering approach described in Section III-D. Detailed documentation for all classes is available on the PCL web site ($www.pointclouds.org$).

To achieve efficient performance, we have parallelized this approach. The planar segmentation approach requires surface normals to have been computed for a given point cloud, so this is performed for the most recent cloud received from the sensor. In parallel with this, planes are segmented for the previous frame, for which normals have already been computed. A system diagram illustrating this processing pipeline is shown in Figure 7.

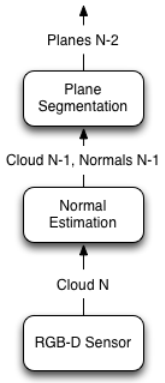

Fig. 7. A system diagram demonstrating our planar segmentation processing pipeline. The normal estimation and plane segmentation operations occur in parallel, enabling real-time performance.

V. Discussion & Evaluation

Our segmentation approach has been applied to several applications, including tabletop object segmentation mapping with planar landmarks. We present quantitative runtime results for several datasets, as well as a qualitative discussion of applications to tabletop object segmentation and SLAM.

A. Runtime Evaluation

The runtime of our approach was evaluated on Kinect data available from the TUM RGB-D Dataset [11] at `http://vision.in.tum.de/data/datasets/rgbd-dataset`. We selected three datasets from this repository with various kinds of scenes. The "freiburg1 desk" dataset contains close-range scenes of an office desktop environment. The "freiburg1 floor" dataset contains scenes of a floor in an indoor environment, so a large plane is present in all frames. The "freiburg2 pioneer_slam" dataset includes scenes from a forward-looking RGB-D camera mounted on a mobile robot moving through a large room, as in a SLAM scenario. These datasets were converted to PCD format, and replayed from files at a rate of 30Hz. These experiments were performed on a 2.6 GHz Intel Core i7 CPU. A multi-threaded evaluation tool using the tools described in PCL implementation from Section IV was designed for this application.

Table I presents runtimes for planar segmentation, as well as the individual steps of normal estimation and planar segmentation. Normal estimation and plane segmentation were run in parallel, which is necessary for real-time performance. However, these steps can also be run sequentially if only a single core is available, at the expense of frame rate. The frame callback timings are the average time elapsed since the previous frame was fully processed.

Fig. 8. An example image from the desk1 dataset, with a colorized representation of the label image superimposed on top. The displayed label image was generated using the planar segmentation approach with refinement.

B. Tabletop Object Segmentation Discussion

The approach described above has been used for tabletop object segmentation from RGB-D camera. An example scene is shown in Figure 1, and a video example is available on YouTube at `http://www.youtube.com/watch?v=Izgy99WHFBs`. Segmented planar surfaces are displayed with colored boundary points, and large red arrows indicating their normal direction. Segmented objects are shown as colored point clouds.

As can be seen, well-separated objects above a planar surface can be segmented in close-range scenes. Highly cluttered scenes pose a challenge, as these tend to be under-segmented. The key to this approach is that using the planar surface as a mask enables the clustering to separate the objects, rather than connecting them via the tabletop surface. In our experience, using the planar refinement step produces a much better mask, leading to a better segmentation.

C. Mapping Discussion

Our segmentation approach has also been used in the context of a 3D mapping task, in which planar surfaces were segmented from an RGB-D camera mounted on a mobile robot moving through an indoor environment. Multiple observations of each surface were merged to create a map such as the one shown in Figure 9. This is similar to the approach using in our previous planar mapping work [12], which was based on RANSAC segmented planes. Qualitatively, we found the planes segmented by the proposed approach to be preferable to the previously used RANSAC based approach. RANSAC necessitated downsampling the data to achieve reasonable performance, which degrades the quality of the planar region boundaries. Additionally, with the RANSAC-based approach, we computed either a convex hull or alpha

	fr1 desk	fr1 floor	fr2 pioneer_slam
Normal Estimation	21.56 ± 2.07ms	22.98 ± 1.93ms	21.97 ± 3.91ms
Plane Segmentation	26.13 ± 3.19ms	21.28 ± 3.09ms	23.17 ± 5.62ms
Frame Callback	33.51 ± 3.13ms	33.62 ± 2.71ms	33.98 ± 4.03ms
Callback Rate	29.83 Hz	29.73 Hz	29.42 Hz

TABLE I

AVERAGE RUNNING TIMES FOR NORMAL ESTIMATION AND PLANE SEGMENTATION WITHOUT PLANAR REFINEMENT.

	fr1 desk	fr1 floor	fr2 pioneer_slam
Normal Estimation	21.91 ± 2.36ms	23.99 ± 2.57ms	22.00 ± 3.86ms
Plane Segmentation+Refinement	32.55 ± 3.29ms	29.83 ± 3.38ms	29.53 ± 6.34ms
Frame Callback	35.79 ± 3.20ms	34.08 ± 3.23ms	35.74 ± 4.35ms
Callback Rate	27.93 Hz	29.34 Hz	27.97 Hz

TABLE II

AVERAGE RUNNING TIMES FOR NORMAL ESTIMATION AND PLANE SEGMENTATION WITH PLANAR REFINEMENT.

shape to represent the boundary, which does not always accurately represent the true shape of the region. In contrast, this approach produces the exact boundary points from the image, producing a better representation of the planar boundary.

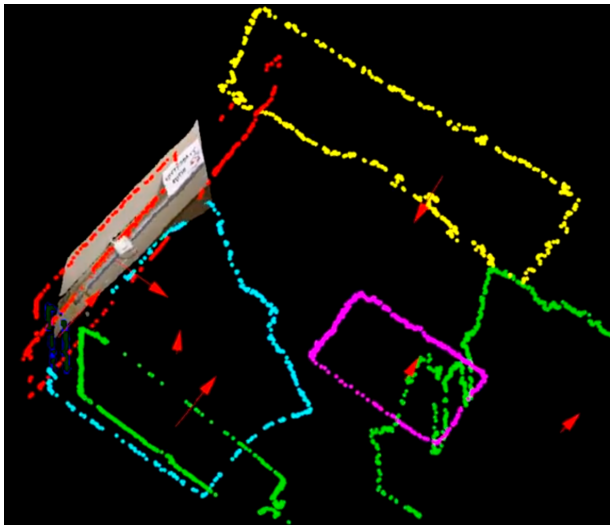

Fig. 9. An example of an indoor mapping tasks using planar landmarks. The colored points represent boundaries of planar surfaces segmented from RGB-D data, and merged from multiple observations.

VI. ACKNOWLEDGMENTS

This work was made possible through financial support from Willow Garage. Thanks to Michael Dixon and Dirk Holz for useful discussions regarding the segmentation approach. We would also like to thank our reviewers for their helpful comments.

REFERENCES

[1] Joydeep Biswas and Manuela Veloso. Depth camera based indoor mobile robot localization and navigation. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1697–1702. IEEE, 2012.

[2] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

[3] Dirk Holz and Sven Behnke. Fast range image segmentation and smoothing using approximate surface reconstruction and region growing. In *Intelligent Autonomous Systems 12*, pages 61–73. Springer, 2013.

[4] Dirk Holz, Stefan Holzer, Radu Bogdan Rusu, and Sven Behnke. Real-time plane segmentation using rgb-d cameras. In *RoboCup 2011: Robot Soccer World Cup XV*, pages 306–317. Springer, 2012.

[5] S Holzer, RB Rusu, M Dixon, S Gedikli, and N Navab. Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 2684–2689. IEEE, 2012.

[6] Rostislav Hulik, Vitezslav Beran, Michal Spanel, Premysl Krsek, and Pavel Smrz. Fast and accurate plane segmentation in depth maps for indoor scenes. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 1665–1670. IEEE, 2012.

[7] Jann Poppinga, Narunas Vaskevicius, Andreas Birk, and Kaustubh Pathak. Fast plane detection and polygonalization in noisy 3d range images. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 3378–3383. IEEE, 2008.

[8] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011.

[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 *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), St. Louis, MO, USA*, 2009.

[10] Linda Shapiro and George C Stockman. *Computer Vision*, chapter 3, pages 69–75. Prentice Hall, 2001.

[11] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[12] A. J. B. Trevor, J. G. Rogers III, and H. I. Christensen. Planar Surface SLAM with 3D and 2D Sensors. *International Conference on Robotics and Automation*, 2012.

[13] Alexander J. B. Trevor, John G Rogers III, Akansel Cosgun, and Henrik I Christensen. Interactive object modeling & labeling for service robots. In *Proceedings of the 8th ACM/IEEE international conference on Human-robot interaction*, pages 421–422. IEEE Press, 2013.

[14] George Vosselman, Ben GH Gorte, George Sithole, and Tahir Rabbani. Recognising structure in laser scanner point clouds. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 46(8):33–38, 2004.