2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

Fast Plane Extraction in Organized Point Clouds Using Agglomerative Hierarchical Clustering

Chen Feng[1], Yuichi Taguchi[2], and Vineet R. Kamat[1]

***Abstract*— Real-time plane extraction in 3D point clouds is crucial to many robotics applications. We present a novel algorithm for reliably detecting multiple planes in real time in organized point clouds obtained from devices such as Kinect sensors. By uniformly dividing such a point cloud into non-overlapping groups of points in the image space, we first construct a graph whose node and edge represent a group of points and their neighborhood respectively. We then perform an agglomerative hierarchical clustering on this graph to systematically merge nodes belonging to the same plane until the plane fitting mean squared error exceeds a threshold. Finally we refine the extracted planes using pixel-wise region growing. Our experiments demonstrate that the proposed algorithm can reliably detect all major planes in the scene at a frame rate of more than 35Hz for 640×480 point clouds, which to the best of our knowledge is much faster than state-of-the-art algorithms.**

I. Introduction

As low-cost depth cameras and 3D sensors have emerged in the market, they have become a popular choice in various robotics and computer vision applications. 3D point clouds obtained by such sensors are generally noisy and redundant, and do not provide semantics of the scene. For compact and semantic modeling of 3D scenes, primitive fitting to the 3D point clouds has attracted a lot of research interests. In particular, planes are one of the most important primitives, since man-made structures mainly consist of planes.

In this paper, we present an efficient plane extraction algorithm applicable to organized point clouds, such as depth maps obtained by Kinect sensors. Our algorithm first constructs a graph by dividing a point cloud into several non-overlapped regions with a uniform size in the image space. The algorithm then performs a bottom-up, agglomerative hierarchical clustering (AHC) on the graph: It repeats (1) finding the region that has the minimum plane fitting mean squared error (MSE) and (2) merging it with one of its neighbors such that the merge results in the minimum plane fitting MSE. We show that the clustering process can be done with the complexity log-linear in the number of initial nodes, enabling real-time plane extraction. To refine the boundaries of the clustered regions, the clustering process is followed by pixel-wise region growing. In experiments, we compare our algorithm with state-of-the-art algorithms. Our algorithm achieves real-time performance (runs over 35 Hz) for 640×480 pixel depth maps, while providing the accuracy comparable to the state-of-the-art algorithms. Some example results are shown in Figure 1.

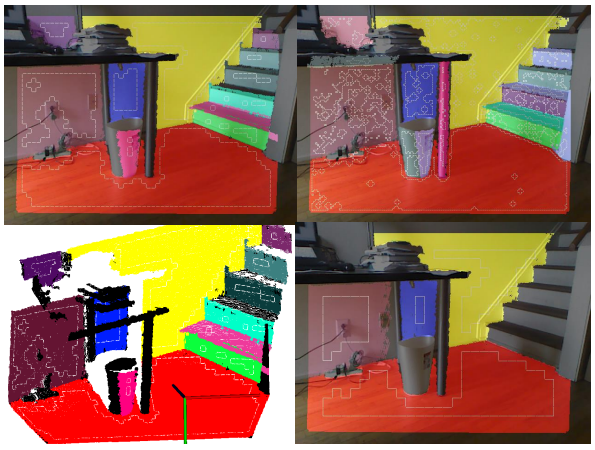

Fig. 1. Plane extraction results generated using our algorithm with different initial node sizes. Extracted planes are superimposed with different colors on the RGB image (black means non-planar region). White dash lines show the segmentation boundaries before the region-grow-based refinement. Initial node size of 10×10 detects most of the planes in the scene (top-left), whose 3D view is shown (bottom-left). Initial node size of 4×4 reveals more segments in a smaller scale such as stairs and table leg (top-right), while that of 20×20 focuses on major large planar structures such as floors and walls (bottom-right).

A. Contributions

This paper makes the following contributions:

- We present an efficient plane extraction algorithm based on agglomerative clustering for organized point clouds.
- We analyze the complexity of the clustering algorithm and show that it is log-linear in the number of initial nodes.
- We demonstrate real-time performance with the accuracy comparable to state-of-the-art algorithms.

B. Related Work

Plane Extraction: Several different algorithms have been proposed for plane extraction from 3D point clouds. RANSAC-based methods [1] have been widely used. These methods usually follow the paradigm of iteratively applying RANSAC algorithm on the data while removing inliers corresponding to the currently found plane instance. Since RANSAC requires relatively long computation time for random plane model selection and comparison, several different variants were developed. Oehler et al. [2] performed Hough transformation and connected component analysis

[1]Chen Feng and Vineet R. Kamat are with the Department of Civil & Environmental Engineering, University of Michigan, Ann Arbor, MI 48109, USA cforrest at umich.edu

[2]Yuichi Taguchi is with Mitsubishi Electric Research Labs, 201 Broadway, Cambridge, MA 02139, USA taguchi at merl.com

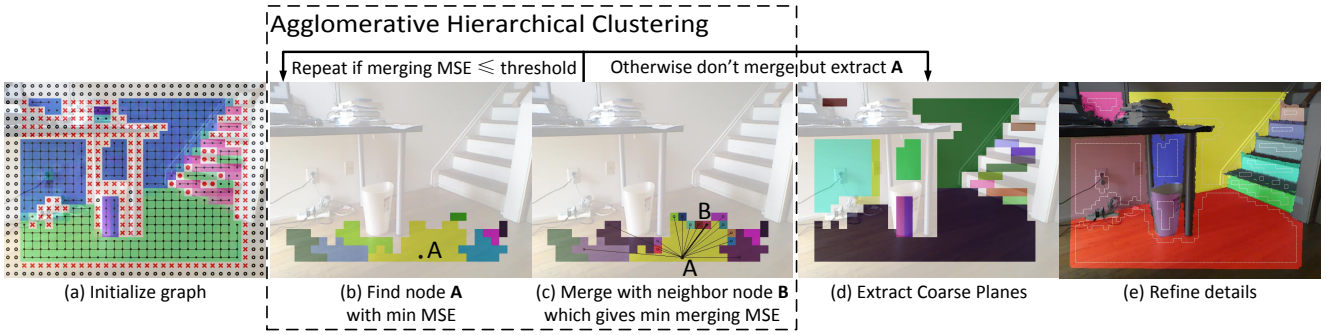

Fig. 2. Algorithm overview. Each frame of an organized point cloud is processed from left to right. (a) shows the graph initialization with each node colored by its normal; black dot and line showing graph node and edge; red 'x', black 'o', and red dot showing node rejected by depth discontinuity, missing data, and too large plane fitting MSE, respectively. (b) and (c) show the two core operations of the AHC. Regions with random colors in (b) and (c) show graph nodes merged at least once. Black lines in (c) show all edges coming out from the node A, in which the thick line shows the edge to the node B that gives the minimum plane fitting MSE when merging the node A with one of its neighbors. Colored regions in (d) show the extracted coarse planes, which are finally refined in (e) if required by the application.

on the point cloud first as pre-segmentation and then applied RANSAC to refine each of the resulting "surfels" (2s per 640×480 points). Several algorithms [3]–[5] applied RANSAC on local regions of the point cloud (which decreases the data size considered in each RANSAC run so as to increase speed) and then grew the region from the locally found plane instance to the whole point cloud (0.2s [3] or 0.1s [4] per 640×480 points; 0.03s [5] per 320×240 points). Region-grow-based methods are another popular choice. Hähnel et al. [6] and Poppinga et al. [7] grew points by both point-plane distance threshold and MSE threshold (0.2s per 25,344 points). Holz et al. [8] grew points by their surface normal deviation (0.5s per 640×480 points), which requires per-point normal estimation. A similar but much slower variant is voxel grow [9]. Instead of growing points, Geogiev et al. [10] first extracted line segments from each scan line of the data and then grew the line segments across scan lines (0.05s per 18,100 points in MATLAB).

There are other methods which do not belong to the two groups. Holz et al. [11] first clustered the point cloud in the normal space and further clustered each group by its distance to the origin (0.14s per 640×480 points). To avoid per-point normal estimation, Enjarini et al. [12] designed the gradient of depth feature for plane segmentation, which could be rapidly computed. Graph-based segmentation using self-adaptive threshold was also used [13], [14] (0.17s per 148,500 points [13]). Although our method also uses a graph to represent data relation, our method differs from the previous methods as follows: 1) no RGB information is used; 2) no per-point normal estimation is required; and more importantly, 3) dynamic edge weights are used instead of static ones which fix the merging order as in [13].

Applications: Planes have been used in various applications in robotics, computer vision, and 3D modeling. Compact and semantic modeling of scenes provided by planes is useful in indoor and outdoor 3D reconstruction, visualization, and Building Information Modeling (BIM) [15]. Extracting a major plane is a common strategy for table-top manipulation [11], because it helps segment objects placed on the plane. Planes have been also used for SLAM [16]–[18] and place recognition [19] systems as landmarks, because planes are more robust to noise and more discriminative than points. However, at least three planes whose normals span $\mathbb{R}^3$ are required to compute the 6-degrees-of-freedom camera pose. To avoid the degeneracy due to the insufficient number of planes, Taguchi et al. [3] used both points and planes as landmarks in their SLAM system. Salas-Moreno et al.'s SLAM system that uses objects as landmarks [20] extracted a ground plane and used it as a soft constraint to align the poses of objects with respect to the ground plane. All of the above works can benefit from fast and accurate plane extraction, which we present in this paper.

II. Algorithm Overview

Figure 2 illustrates how our algorithm processes each frame of an organized point cloud. We define an organized point cloud to be a set of 2D indexed 3D points $\mathcal{F} = \{\boldsymbol{p}_{i,j} = (x_{i,j}, y_{i,j}, z_{i,j})^\mathsf{T}\}, i = 1, \cdots, \mathsf{M}, j = 1, \cdots, \mathsf{N}$, where the 2D indices (i, j) and $(i \pm 1, j \pm 1)$ reflect the 3D proximity relationship between points $\boldsymbol{p}_{i,j}$ and $\boldsymbol{p}_{i\pm1,j\pm1}$ if they lie on the same surface (we call this index space as image space). Usually it can be obtained from a depth map produced by devices such as a Kinect sensor, time-of-flight camera, structured light scanning system, and even rotating the scanning plane of a laser range finder.

A. Line Segment Extraction as an Analogy

Before moving into the details of our algorithm, we briefly discuss a line segment extraction algorithm called *line regression*, as summarized in [21] and implemented in April Robotics Toolkit [22]. It is widely used for extracting line features from 2D point sequences obtained from a laser range finder, and inspired us to generalize its idea to 3D case for fast plane extraction. As illustrated in Figure 3, every W consecutive points (W = 3 in this figure) in the sequence are grouped into nodes[1], forming a double linked list. Then

[1]We use "node" and "segment" interchangeably to represent a set of data points.

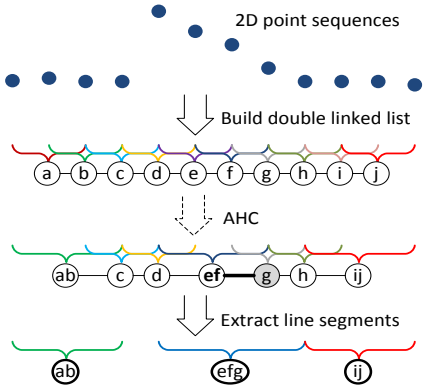

Fig. 3. Line regression algorithm. Blue dots show the 2D points. Circles labeled with letters show the nodes in a linked list. Brackets show the groups of points represented by the nodes. Thick line indicates that merging node g with its left neighbor ef gives a smaller line fitting MSE than merging it with its right neighbor h.

AHC is performed on this linked list by repeating (1) finding the node g with the minimum line fitting MSE and (2) merging this node g with either its left or right neighbor that gives the minimum merging MSE. If the minimum merging MSE is larger than a predefined threshold, which can usually be decided by the noise characteristics of the sensor, then the merging is canceled and the node g can be extracted as a line segment. When using a binary heap to find the minimum MSE node, log-linear time complexity $O(n \log n)$ can be achieved for this algorithm, where n is the number of points in the sequence. Note that by applying the idea of integral images, as used in [11], [23], merging two nodes and calculating the resulting line fitting MSE become constant time operations.

B. Differences When Generalizing to 3D

Inspired by the use of point's neighborhood information given by the point's order of the sequence, we wish to generalize the 2D line regression to 3D plane extraction in an organized point cloud, where the neighborhood information is stored in the 2D indices. However, this generalization is nontrivial, because of the following two major differences.

Non-Overlapping Nodes: As opposed to the line regression, initial nodes (and thus any two nodes during/after merging) should have no identical points, i.e., for any two nodes $\mathcal{B}_s, \mathcal{B}_t \subset \mathcal{F}$, $\mathcal{B}_s \cap \mathcal{B}_t = \varnothing$. This requirement is due to the fact that after several merging steps, the 3D points belonging to a certain node $\mathcal{B}_s$ will form an irregular shape instead of maintaining its initial rectangular shape in the image space, as shown in Figure 2(b). Thus, if allowing different nodes to have identical points, it is difficult to efficiently handle the overlapping points when merging two nodes, even with the help of integral images. While in the line regression, merging two neighboring line segments will still result in a line segment represented by a start and end index in the point sequence, which makes overlapping nodes feasible. It is important to notice that the overlapping nodes enable the line regression algorithm to automatically split line segments at their boundaries; since nodes containing points at different line segments tend to have larger line fitting MSE than others (e.g., nodes c, d, and h in Figure 3), their merging attempts will be delayed and finally rejected. The non-overlapping requirement in our algorithm results in losing that advantage of automatically detecting boundaries of planes. We will describe how to overcome the disadvantage by removing bad nodes in the initialization step in Section III-A. We will also describe a pixel-wise region growing algorithm to refine the boundaries of planes in Section IV.

Algorithm 1 Fast Plane Extraction

1: **function** FASTPLANEEXTRACTION($\mathcal{F}$)
2: $G \leftarrow$ INITGRAPH($\mathcal{F}$)
3: $(\mathcal{B}, \Pi) \leftarrow$ AHCLUSTER(G)
4: $(\mathcal{C}, \Pi') \leftarrow$ REFINE($\mathcal{B}, \Pi$)
5: **return** $(\mathcal{C}, \Pi')$

Number of Merging Attempts: In the line regression, merging a node with its neighbor is a constant time operation with at most two merging attempts, either to its left or right neighbor. In our case, the number of merging attempts is larger, since nodes are initially connected to at most 4 neighbors to form a graph, and after several merging steps, they can be connected to a larger number of neighbors. In Section III-B, we will experimentally analyze the average number of merging attempts in our algorithm and show that it stays small in practice; therefore, the merging step can be done in a constant time, resulting in the complexity of $O(n \log n)$ similar to the line regression.

III. Fast Coarse Segmentation

Our fast plane extraction algorithm consists of three major steps, as shown in Figure 2 and Algorithm 1: The algorithm first initializes a graph and then performs AHC for extracting coarse planes, which are finally refined. If the application only requires rough segmentation of planar regions, e.g., detecting objects in a point cloud, then the final refinement step may be skipped, which could increase the frame rate to more than 50Hz for 640×480 points.

First we clarify our notations. $\mathcal{F}$ denotes a complete frame of an organized point cloud of M rows and N columns. $\mathcal{B}, \mathcal{C}$ represent coarse and refined segmentation respectively, i.e., each element $\mathcal{B}_k/\mathcal{C}_l$ of $\mathcal{B}/\mathcal{C}$ is a segment—a set of 3D points $\boldsymbol{p}_{i,j}$. Meanwhile Π, Π' are sets of plane equations corresponding to $\mathcal{B}, \mathcal{C}$, respectively. Also note that each node v of a graph G is a set of 3D points and each undirected edge uv denotes the neighborhood of segments u, v in the image space.

A. Graph Initialization

As mentioned in Section II-B, our algorithm has a requirement of non-overlapping node initialization, represented in lines 3 to 5 of Algorithm 2. This step uniformly divides the point cloud into a set of initial nodes of the size $\mathsf{H} \times \mathsf{W}$ in the image space. The requirement causes our algorithm to lose the advantage of automatically detecting boundaries

Algorithm 2 Graph Initialization

function INITGRAPH($\mathcal{F}$)
  $G \leftarrow (V \leftarrow \varnothing, E \leftarrow \varnothing)$
  for $i \leftarrow 1, \lceil \frac{\mathsf{M}}{\mathsf{H}} \rceil$ **do** ▷ initialize nodes
    for $j \leftarrow 1, \lceil \frac{\mathsf{N}}{\mathsf{W}} \rceil$ **do**
      $v_{i,j} \leftarrow \{\boldsymbol{p}_{k,l}\} \subset \mathcal{F}, k = (i-1)\mathsf{H}+1, \cdots, \min(i\mathsf{H}, \mathsf{M}), l = (j-1)\mathsf{W}+1, \cdots, \min(j\mathsf{W}, \mathsf{N})$
      if REJECTNODE($v_{i,j}$) **then**
        $v_{i,j} \leftarrow \varnothing$
      $V \leftarrow V \cup \{v_{i,j}\}$
  for each $v_{i,j} \in V$ **do** ▷ initialize edges
    if $\neg$ REJECTEDGE($v_{i,j-1}, v_{i,j}, v_{i,j+1}$) **then**
      $E \leftarrow E \cup \{v_{i,j-1}v_{i,j}, v_{i,j}v_{i,j+1}\}$
    if $\neg$ REJECTEDGE($v_{i-1,j}, v_{i,j}, v_{i+1,j}$) **then**
      $E \leftarrow E \cup \{v_{i-1,j}v_{i,j}, v_{i,j}v_{i+1,j}\}$
  return G

function REJECTNODE(v)
  if v contains missing data point **then return true**
  else if any point $\boldsymbol{p}_{i,j} \in v$ is depth-discontinuous with any of its 4 neighbor points **then return true**
  else if MSE(v) $> \mathsf{T}_{\mathsf{MSE}}$ **then return true**
  else return false

function REJECTEDGE(v_a, v_b, v_c)
  if $v_a = \varnothing \vee v_b = \varnothing \vee v_c = \varnothing$ **then return true**
  else if included angle of plane fitting normal of v_a and v_c is greater than $\mathsf{T}_{\mathsf{ANG}}$ **then return true**
  else return false

function MSE(v)
  if $v = \varnothing$ **then return** $+\infty$
  else return the plane fitting MSE for all $\boldsymbol{p}_{i,j} \in v$

of planes. To properly segment planes using AHC under this restriction, we remove the following types of nodes and corresponding edges from the graph, which are illustrated using an example in Figure 4:

1) **Nodes Having High MSE:** Non-planar regions lead to high plane fitting MSE, which we simply remove.
2) **Nodes Containing Missing Data:** Because of the limitation of the sensor, some regions of the scene might not be sensed correctly, leading to missing data (e.g., the glass window behind the shutter).
3) **Nodes Containing Depth Discontinuities:** These nodes contain two sets of points lying on two surfaces that are not close in 3D but are close in the image space (usually one surface partially occludes the other, e.g., the monitor occludes the wall behind). If principle component analysis (PCA) is performed on points belonging to this node for plane fitting, the fitted plane will be nearly parallel to the line-of-sight direction and thus still have a small MSE. Merging this "outlier" node with its neighbor node will have bad effect on the plane fitting result because of the well-known issue of over-weighting outliers in least-squares methods.
4) **Nodes at Boundary Between Two Planes:** These nodes contain two sets of points close to each other in 3D but lying on two difference planes (e.g., the corner of the room), which will decrease the plane fitting accuracy if they are merged to one of the planes.

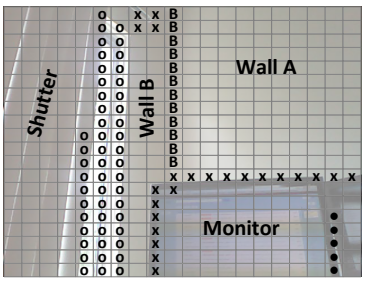

Fig. 4. Examples of bad initial nodes. 'o' shows nodes with missing data point; 'x' shows nodes with depth discontinuity; black dot shows nodes with too large plane fitting MSE; and 'B' shows nodes located at the boundary region between two connected planes.

The functions REJECTNODE and REJECTEDGE in Algorithm 2 are designed to reduce the influence of these four types of bad initial nodes. The REJECTNODE function removes the first three types of bad nodes (and thus the points inside) from the graph, while the REJECTEDGE function is for mitigating influence of the fourth type of bad nodes.

It is interesting to note that the gain in this non-overlapping "disadvantage" is the avoidance of per-point normal estimation. Our initialization step can be seen as treating all points inside a node as if they have a common plane normal. This is an important reason for our speed improvement compared to other state-of-the-art methods which often spend a large portion of time in the normal estimation for each point.

B. Agglomerative Hierarchical Clustering

As shown in Algorithm 3, the AHC in our algorithm is almost the same as that in the line regression, except that it is operated on a graph instead of a double linked list. We first build a min-heap data structure for efficiently finding the node with the minimum plane fitting MSE. We then repeat finding a node v that currently has the minimum plane fitting MSE among all nodes in the graph and merging it with one of its neighbor nodes u_{best} that results in the minimum merging MSE (recall that each node in the graph is a set of points; so the merging MSE is the plane fitting MSE of the union of the two sets u_{merge}). If this minimum merging MSE exceeds some predefined threshold $\mathsf{T}_{\mathsf{MSE}}$ (not necessarily a fixed parameter as explained later in Section III-C), then a plane segment v is found and extracted from the graph; otherwise the merged node u_{merge} is added back to the graph by edge contraction between v and u_{best}.

As mentioned in Section II-B, our algorithm requires a larger number of merging attempts than the line regression. However, it turns out to be still quite efficient and the clustering process can be done in $O(n \log n)$ time in

Algorithm 3 Agglomerative Hierarchical Clustering

1: **function** AHCLUSTER($G = (V, E)$)
2: $Q \leftarrow$BUILDMINMSEHEAP(V)
3: $\mathcal{B} \leftarrow \varnothing, \Pi \leftarrow \varnothing$
4: **while** $Q \neq \varnothing$ **do**
5: $v \leftarrow$POPMIN(Q)
6: **if** $v \notin V$ **then** ▷ v was merged previously
7: **continue**
8: $u_{best} \leftarrow \varnothing, u_{merge} \leftarrow \varnothing$
9: **for each** $u \in N(v) \triangleq \{u | uv \in E\}$ **do**
10: $u_{test} \leftarrow u \cup v$ ▷ merge attempt
11: **if** MSE(u_{test}) $<$ MSE(u_{merge}) **then**
12: $u_{best} \leftarrow u, u_{merge} \leftarrow u_{test}$
13: **if** MSE(u_{merge}) $\geqslant \mathsf{T}_{\mathsf{MSE}}$ **then** ▷ merge fail
14: **if** $|v| \geqslant \mathsf{T}_{\mathsf{NUM}}$ **then** ▷ extract node v
15: $\mathcal{B} \leftarrow \mathcal{B} \cup \{v\}, \Pi \leftarrow \Pi \cup$PLANE($v$)
16: $E \leftarrow E \setminus E(v) \triangleq \{uv | u \in N(v)\}$
17: $V \leftarrow V \setminus \{v\}$ ▷ reject small node
18: **else** ▷ merge success
19: INSERT(Q, u_{merge})
20: $E \leftarrow E \cup \{u_{merge} w | w \in N(v) \cup N(u_{best}) \setminus \{v, u_{best}\}\} \setminus E(u_{best}) \setminus E(v)$ ▷ edge contraction
21: $V \leftarrow V \cup \{u_{merge}\} \setminus \{v, u_{best}\}$
22: **return** $(\mathcal{B}, \Pi)$

23: **function** PLANE(v)
24: **return** plane equation fitted from points in v by PCA

practice. Figure 5 experimentally shows the average number of merging attempts during AHC per frame. As can be seen, irrespective of the initial node size (and thus the initial number of nodes), this number stays small. This may be explained by the fact that the graph constructed from Algorithm 2 is a planar graph. From graph theory one knows that the average node degree of a planar graph is strictly less than 6. Since our initial graph is planar and merging nodes by edge contraction does not change its planarity, during the whole process of AHC the average node degree is always less than 6. Also, the plane fitting MSE of a large segment is larger than that of a smaller segment, if errors are drawn from the same Gaussian distribution. Thus the AHC process tends to balance the size of all the segments, because it always tries to grow the size of the node with the minimum plane fitting MSE and then switches to other smaller nodes. Therefore, it will not stick to growing a large node (which implies large node degree since it has large boundary), otherwise the average number of merging tests will be much larger. Based on this observation, lines 6 to 21 in Algorithm 3 can be done in a constant time irrespective of the initial number of nodes. The $O(n \log n)$ complexity only arises from maintaining the min-heap structure.

C. Implementation Details

There are several implementation details to improve the speed and accuracy for this fast coarse segmentation:

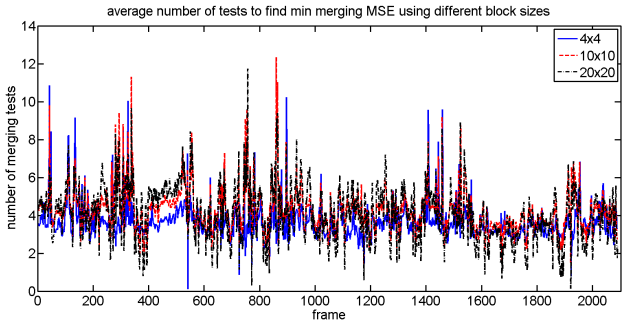

Fig. 5. Average number of merging tests per frame during 2102 frames of 640×480 Kinect point clouds. Three initial node sizes are tested.

1) A disjoint set data structure is used for tracking the point membership of each initial node $v_{i,j}$.
2) As in the line regression, all nodes maintain the first and second order statistics of all the belonging points, i.e.,
$$\sum x_{i,j}, \sum y_{i,j}, \sum z_{i,j}, \sum x_{i,j}^2, \sum y_{i,j}^2, \sum z_{i,j}^2, \sum x_{i,j} y_{i,j}, \sum y_{i,j} z_{i,j}, \sum z_{i,j} x_{i,j},$$
such that merging two nodes and calculating its plane equation and MSE through PCA is a constant time operation.
3) The function for determining the depth discontinuity in REJECTNODE of Algorithm 2 depends on sensor noise characteristics. For Kinect sensors, we use the following function as suggested in [23] and Point Cloud Library (PCL)[2]:
$$f(\boldsymbol{p}_a, \boldsymbol{p}_b) = \begin{cases} 1 & |z_a - z_b| > 2\alpha(|z_a| + 0.5) \\ 0 & \text{otherwise} \end{cases} \quad (1)$$
The unit of z here (and throughout the paper) is millimeter and the parameter α we used was between 0.01 and 0.02.
4) The threshold $\mathsf{T}_{\mathsf{MSE}}$ for extracting segments is also sensor dependent. For Kinect, we use the following equation adapted from [24]
$$\mathsf{T}_{\mathsf{MSE}} = (\sigma z^2 + \epsilon)^2, \quad (2)$$
where we used $\sigma = 1.6 \times 10^{-6}$ and ϵ between 3 and 8. Similarly, $\mathsf{T}_{\mathsf{ANG}}$ can also be changed depending on depth.
5) The initial node should be close to a square shape in the image space, i.e., $\mathsf{W} \approx \mathsf{H}$. If a strip-like shape is used, either $\mathsf{W} \gg \mathsf{H}$ (e.g., $\mathsf{W} = 20, \mathsf{H} = 2$) or $\mathsf{H} \gg \mathsf{W}$, the PCA on the initial node will result in wrong plane normal direction which is usually almost perpendicular to the line-of-sight direction. Consequently the following AHC will fail to segment planes correctly.

IV. SEGMENTATION REFINEMENT

For many applications, the coarse plane segmentation obtained in the previous section might not be enough, especially

[2] http://www.pointclouds.org/

Algorithm 4 Segmentation Refinement

1: **function** REFINE($\mathcal{B}, \Pi$)
2: $Q \leftarrow \varnothing$ ▷ initialize queue for boundary points
3: $\mathcal{R} \leftarrow \varnothing$ ▷ points to be refined
4: $G' \leftarrow (V' \leftarrow \varnothing, E' \leftarrow \varnothing)$ ▷ graph for final merge
5: **for each** $\mathcal{B}_k \in \mathcal{B}$ **do** ▷ 1. erode each segment
6: $\mathcal{R}_k \leftarrow \varnothing, \mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}_k$
7: **for each** initial node $v_{i,j} \subset \mathcal{B}_k$ **do**
8: **if** $v_{i-1,j} \cup v_{i+1,j} \cup v_{i,j-1} \cup v_{i,j+1} \not\subset \mathcal{B}_k$ **then**
9: $\mathcal{B}_k \leftarrow \mathcal{B}_k \setminus v_{i,j}$ ▷ erode border node
10: **for each** point $\boldsymbol{p}_{s,t}$ on the boundary of $\mathcal{B}_k$ **do**
11: ENQUEUE($Q, (\boldsymbol{p}_{s,t}, k)$)
12: **if** $\mathcal{B}_k \neq \varnothing$ **then**
13: $V' \leftarrow V' \cup \{\mathcal{B}_k\}$
14: **while** $Q \neq \varnothing$ **do** ▷ 2. region grow from boundary
15: $(\boldsymbol{p}_{s,t}, k) \leftarrow$ DEQUEUE(Q)
16: **for** $\boldsymbol{p}_{i,j} \in \{\boldsymbol{p}_{s-1,t}, \boldsymbol{p}_{s+1,t}, \boldsymbol{p}_{s,t-1}, \boldsymbol{p}_{s,t+1}\}$ **do**
17: **if** $\boldsymbol{p}_{i,j} \in (\mathcal{B}_k \cup \mathcal{R}_k) \vee$ DIST$(\boldsymbol{p}_{i,j}, \Pi_k)^2 \geqslant 9 \cdot$MSE$(\mathcal{B}_k)$ **then continue**
18: **if** $\exists l, \boldsymbol{p}_{i,j} \in \mathcal{R}_l$ **then**
19: $E' \leftarrow E' \cup \{\mathcal{B}_k \mathcal{B}_l\}$ ▷ connect nodes
20: **if** DIST$(\boldsymbol{p}_{i,j}, \Pi_k) <$ DIST$(\boldsymbol{p}_{i,j}, \Pi_l)$ **then**
21: $\mathcal{R}_l \leftarrow \mathcal{R}_l \setminus \{\boldsymbol{p}_{i,j}\}, \mathcal{R}_k \leftarrow \mathcal{R}_k \cup \{\boldsymbol{p}_{i,j}\}$
22: ENQUEUE($Q, (\boldsymbol{p}_{i,j}, k)$)
23: **else**
24: $\mathcal{R}_k \leftarrow \mathcal{R}_k \cup \{\boldsymbol{p}_{i,j}\}$
25: ENQUEUE($Q, (\boldsymbol{p}_{i,j}, k)$)
26: **for each** $\mathcal{R}_k \in \mathcal{R}$ **do**
27: $\mathcal{B}_k \leftarrow \mathcal{B}_k \cup \mathcal{R}_k$ ▷ update each coarse segment
28: $(\mathcal{C}, \Pi') \leftarrow$AHCLUSTER($G'$) ▷ 3. final merge
29: **return** $(\mathcal{C}, \Pi')$

if the applications use the boundaries of planes or require higher accuracy of the estimated plane equations. Thus we perform refinement on the coarse segmentation $\mathcal{B}$.

Three types of artifacts are expected in the coarse segmentation, as shown in Figure 6:

- **Sawtooth:** Usually at the boundary between two connected planes.
- **Unused Data Points:** Usually at the boundary of occlusion or missing data node.
- **Over-Segmentation:** Usually between two object's occlusion boundary.

Sawtooth artifacts cause small amount of outliers to be included in estimation, whereas unused data points and over-segmentation cause less inliers to be used. All of the artifacts produce inaccurate plane boundaries and slightly decrease the accuracy of the estimated plane equation.

Our solution to them is described in Algorithm 4. Since sawtooth artifacts are almost always observed at the boundary regions of $\mathcal{B}$, erosion of boundary regions of each segment can effectively eliminate them (lines 5 to 13). Then pixel-wise region growing is started from all new boundary points to assign all unused data points to its closest plane that is extracted previously (lines 14 to 27). During the region growing the 4-connected neighborhoods are discovered for each segment $\mathcal{B}_k$, which form a new graph G'. Finally applying AHC again on this very small graph (usually less than 30 nodes) fixes the over-segmentation artifact (line 28).

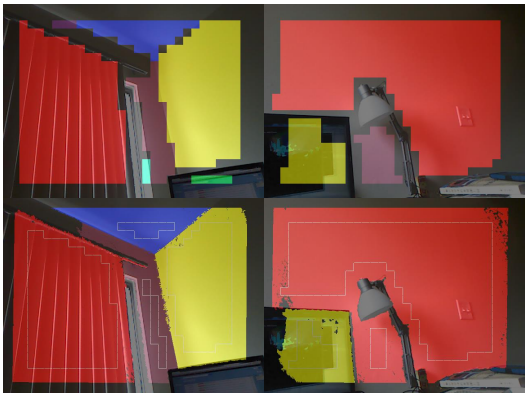

Fig. 6. Top row shows several artifacts in coarse segmentations. Top-left: Sawtooth (e.g., purple and yellow segments). Top-right: Unused data points (e.g., between lamp and wall) and over-segmentation (e.g., purple and red segments). Bottom row shows the corresponding refined segmentations.

V. EXPERIMENTS AND DISCUSSION

To comprehensively evaluate our algorithm's performance in terms of robustness, time, and accuracy, we conducted three sets of experiments described in the following subsections. We implemented our algorithm in C/C++. For PCA, we used the efficient 3×3 matrix eigenvalue decomposition routine described in [25][3]. All experiments were conducted on an ordinary laptop with Intel Core i7-2760QM CPU of 2.4GHz and RAM of 8GB. No multi-threading or any other parallelism such as OpenMP or GPU was used in our implementation.

A. Simulated Data

Similar to the influence of noise simulation in [10], we tested our algorithm's robustness on a simulated depth map with 20 different levels of uniformly distributed noise of magnitude $E = 10l, l = 0, \ldots, 20$ (noise unit: mm; ground truth depth ranges from 1396mm to 3704mm). After the noise is added to the depth map, we converted it to an organized point cloud and fed into our algorithm ($\mathsf{W} = \mathsf{H} = 20, \mathsf{T}_{\mathsf{MSE}} = 50^2$). As shown in Figure 7, our algorithm can reliably detect all of the 4 planes for $l = 0, \ldots, 14$, and starts to over-segment after that. Yet even when $E = 200$ mm our algorithm was able to detect major planes in the scene.

B. Real-World Kinect Data

To measure the processing speed of our algorithm, 2102 frames of 640×480 pixel real-world Kinect data were collected in an indoor scene, partly shown in Figures 1 and 6. Then they were processed with our algorithm using 12 different initial node sizes ($\mathsf{T}_{\mathsf{NUM}} = 800, \alpha = 0.02, \epsilon =$

[3]Implementation available for download at http://www.mpi-hd.mpg.de/personalhomes/globes/3x3/

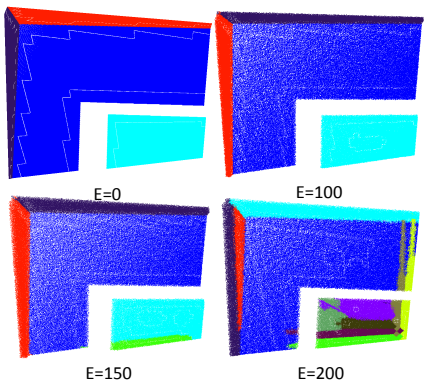

Fig. 7. Plane extraction results on simulated data.

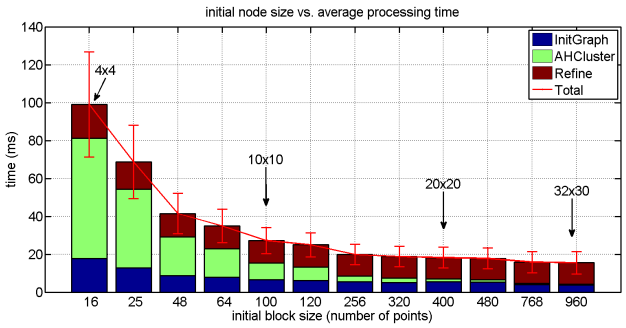

Fig. 8. Average processing time over 2102 frames of 640×480 pixel Kinect data using different initial node sizes.

8mm,$\mathsf{T}_{\mathsf{ANG}}$ increases linearly from 15° at $z = 500$mm to 90° at $z = 4000$mm). As shown in Figure 8, with initial node size of 10×10, even with refinement, our algorithm took only $\mathbf{27.3 \pm 6.9ms}$ in average to process a frame of 640×480 pixel Kinect data, achieving more than **35Hz** frame rate. To the best of our knowledge, this is much faster than other state-of-the-art algorithms.

C. SegComp Datasets

We evaluated the accuracy of our algorithm using the SegComp datasets [26]. Both the ABW ($\mathsf{W} = \mathsf{H} = 4, \mathsf{T}_{\mathsf{MSE}} = 1, \mathsf{T}_{\mathsf{ANG}} = 60^\circ, \mathsf{T}_{\mathsf{NUM}} = 160, \alpha = 0.1$) and PERCEPTRON ($\mathsf{W} = \mathsf{H} = 8, \mathsf{T}_{\mathsf{MSE}} = 2.1, \mathsf{T}_{\mathsf{ANG}} = 45^\circ, \mathsf{T}_{\mathsf{NUM}} = 240, \alpha = 0.03$) datasets of planar scenes were experimented. Typical segmentation results of ABW and PERCEPTRON test datasets are shown in Figure 9. The detailed benchmark results using the evaluation tool provided by SegComp are shown in Table V-C. As can be seen, our algorithm's performance is comparable to the state-of-the-art in terms of segmentation accuracy as well as plane orientation estimation, especially considering the fact that our frame rate is much higher.

VI. Conclusions

We presented a novel fast plane extraction algorithm for organized point clouds, achieving more than 35Hz frame rate on 640×480 point clouds while providing accurate segmentation. In the future we wish to extend the algorithm to non-organized point clouds as well as to fast extraction of other primitives such as spheres and cylinders.

Acknowledgment

We thank Jay Thornton and Srikumar Ramalingam for valuable discussion. This work was supported by Mitsubishi Electric Research Labs.

References

[1] R. Schnabel, R. Wahl, and R. Klein, "Efficient RANSAC for point-cloud shape detection," *Computer Graphics Forum*, vol. 26, no. 2, pp. 214–226, June 2007.

[2] B. Oehler, J. Stueckler, J. Welle, D. Schulz, and S. Behnke, "Efficient multi-resolution plane segmentation of 3D point clouds," in *Proc. Int'l Conf. Intelligent Robotics and Applications (ICIRA)*, Dec. 2011, pp. 145–156.

[3] Y. Taguchi, Y.-D. Jian, S. Ramalingam, and C. Feng, "Point-plane SLAM for hand-held 3D sensors," in *Proc. IEEE Int'l Conf. Robotics and Automation (ICRA)*, May 2013, pp. 5182–5189.

[4] R. Hulik, V. Beran, M. Spanel, P. Krsek, and P. Smrz, "Fast and accurate plane segmentation in depth maps for indoor scenes," in *Proc. IEEE/RSJ Int'l Conf. Intelligent Robots and Systems (IROS)*, Oct. 2012, pp. 1665–1670.

[5] T. Lee, S. Lim, S. Lee, S. An, and S. Oh, "Indoor mapping using planes extracted from noisy RGB-D sensors," in *Proc. IEEE/RSJ Int'l Conf. Intelligent Robots and Systems (IROS)*, Oct. 2012, pp. 1727–1733.

[6] D. Hähnel, W. Burgard, and S. Thrun, "Learning compact 3D models of indoor and outdoor environments with a mobile robot," *Robotics and Autonomous Systems*, vol. 44, no. 1, pp. 15–27, 2003.

[7] J. Poppinga, N. Vaskevicius, A. Birk, and K. Pathak, "Fast plane detection and polygonalization in noisy 3D range images," in *Proc. IEEE/RSJ Int'l Conf. Intelligent Robots and Systems (IROS)*, Sept. 2008, pp. 3378–3383.

[8] D. Holz and S. Behnke, "Fast range image segmentation and smoothing using approximate surface reconstruction and region growing," in *Proc. Int'l Conf. Intelligent Autonomous Systems (IAS)*, June 2012, pp. 61–73.

[9] J.-E. Deschaud and F. Goulette, "A fast and accurate plane detection algorithm for large noisy point clouds using filtered normals and voxel growing," in *Proc. Int'l Symp. 3D Data Processing, Visualization, and Transmission (3DPVT)*, 2010.

[10] K. Georgiev, R. T. Creed, and R. Lakaemper, "Fast plane extraction in 3D range data based on line segments," in *Proc. IEEE/RSJ Int'l Conf. Intelligent Robots and Systems (IROS)*, Sept. 2011, pp. 3808–3815.

[11] D. Holz, S. Holzer, R. B. Rusu, and S. Behnke, "Real-time plane segmentation using RGB-D cameras," in *Proc. RoboCup Symposium*, 2011, pp. 306–317.

[12] B. Enjarini and A. Graser, "Planar segmentation from depth images using gradient of depth feature," in *Proc. IEEE/RSJ Int'l Conf. Intelligent Robots and Systems (IROS)*, Oct. 2012, pp. 4668–4674.

[13] J. Strom, A. Richardson, and E. Olson, "Graph-based segmentation for colored 3D laser point clouds," in *Proc. IEEE/RSJ Int'l Conf. Intelligent Robots and Systems (IROS)*, Oct. 2010, pp. 2131–2136.

[14] T. Wang, L. Chen, and Q. Chen, "A graph-based plane segmentation approach for noisy point clouds," in *Proc. Chinese Control and Decision Conference (CCDC)*, May 2013, pp. 3770–3775.

[15] G. Zhang, P. Karasev, I. Brilakis, and P. Vela, "A sparsity-inducing optimization algorithm for the extraction of planar structures in noisy point-cloud data," in *Proc. Computing in Civil Engineering*, June 2012, pp. 317–324.

[16] J. Weingarten and R. Siegwart, "3D SLAM using planar segments," in *Proc. IEEE/RSJ Int'l Conf. Intelligent Robots and Systems (IROS)*, Oct. 2006, pp. 3062–3067.

[17] K. Pathak, A. Birk, N. Vaškevičius, and J. Poppinga, "Fast registration based on noisy planes with unknown correspondences for 3-D mapping," *IEEE Trans. Robotics*, vol. 26, no. 3, pp. 424–441, June 2010.

[18] A. J. B. Trevor, J. G. Rogers III, and H. I. Christensen, "Planar surface SLAM with 3D and 2D sensors," in *Proc. IEEE Int'l Conf. Robotics and Automation (ICRA)*, May 2012, pp. 3041–3048.

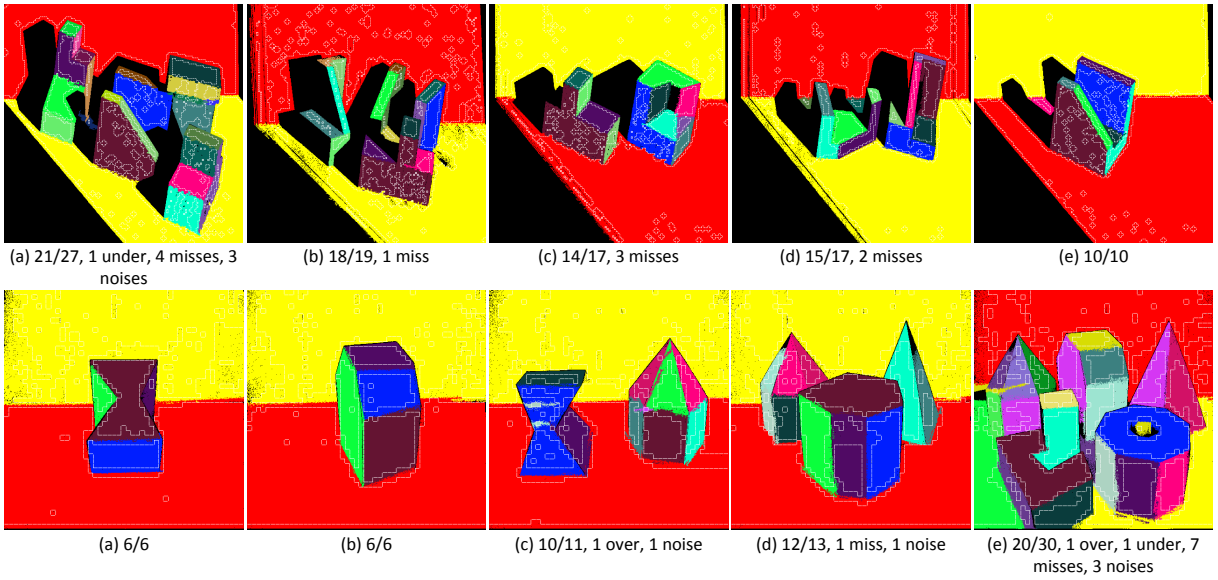

Fig. 9. Plane extraction on SegComp datasets. The estimated plane normal deviated from the ground truth was $(1.7 \pm 0.1)^\circ$ for ABW-TEST (top) and $(2.4 \pm 1.2)^\circ$ for PERCEPTRON-TEST (bottom). Again, white dash lines are the segmentation boundary before the region-grow-based refinement.

TABLE I

BENCHMARKING RESULTS ON THE SEGCOMP DATASETS. THE RESULTS OTHER THAN OURS WERE OBTAINED FROM [2], [8], [27].

Approach	Regions in ground truth	Correctly detected	Orientation deviation (°)	Over-segmented	Under-segmented	Missed (not detected)	Noise (non-existent)
SegComp ABW data set (30 test images) by Hoover et al. [26], assuming 80% pixel overlap as in [27]							
USF [27]	15.2	12.7 (83.5%)	1.6	0.2	0.1	2.1	1.2
WSU [27]	15.2	9.7 (63.8%)	1.6	0.5	0.2	4.5	2.2
UB [27]	15.2	12.8 (84.2%)	1.3	0.5	0.1	1.7	2.1
UE [27]	15.2	13.4 (88.1%)	1.6	0.4	0.2	1.1	0.8
OU [27]	15.2	9.8 (64.4%)	–	0.2	0.4	4.4	3.2
PPU [27]	15.2	6.8 (44.7%)	–	0.1	2.1	3.4	2.0
UA [27]	15.2	4.9 (32.2%)	–	0.3	2.2	3.6	3.2
UFPR [27]	15.2	13.0 (85.5%)	1.5	0.5	0.1	1.6	1.4
Oehler et al. [2]	15.2	11.1 (73.0%)	1.4	0.2	0.7	2.2	0.8
Holz et al. [8]	15.2	12.2 (80.1%)	1.9	1.8	0.1	0.9	1.3
Ours	**15.2**	**12.8 (84.2%)**	**1.7**	**0.1**	**0.0**	**2.4**	**0.7**
SegComp PERCEPTRON data set (30 test images) by Hoover et al. [26], assuming 80% pixel overlap as in [27]							
USF [27]	14.6	8.9 (60.9%)	2.7	0.4	0.0	5.3	3.6
WSU [27]	14.6	5.9 (40.4%)	3.3	0.5	0.6	6.7	4.8
UB [27]	14.6	9.6 (65.7%)	3.1	0.6	0.1	4.2	2.8
UE [27]	14.6	10.0 (68.4%)	2.6	0.2	0.3	3.8	2.1
UFPR [27]	14.6	11.0 (75.3%)	2.5	0.3	0.1	3.0	2.5
Oehler et al. [2]	14.6	7.4 (50.1%)	5.2	0.3	0.4	6.2	3.9
Holz et al. [8]	14.6	11.0 (75.3%)	2.6	0.4	0.2	2.7	0.3
Ours	**14.6**	**8.9 (60.9%)**	**2.4**	**0.2**	**0.2**	**5.1**	**2.1**

[19] E. Fernández-Moral, W. Mayol-Cuevas, V. Arévalo, and J. González-Jiménez, "Fast place recognition with plane-based maps," in *Proc. IEEE Int'l Conf. Robotics and Automation (ICRA)*, May 2013, pp. 2719–2724.

[20] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. J. Kelly, and A. J. Davison, "SLAM++: Simultaneous localisation and mapping at the level of objects," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, June 2013, pp. 1352–1359.

[21] V. Nguyen, A. Martinelli, N. Tomatis, and R. Siegwart, "A comparison of line extraction algorithms using 2D laser rangefinder for indoor mobile robotics," in *Proc. IEEE/RSJ Int'l Conf. Intelligent Robots and Systems (IROS)*, Aug. 2005, pp. 1929–1934.

[22] E. Olson, "The APRIL robotics toolkit," http://april.eecs.umich.edu/wiki/index.php/Main_Page\#April_Robotics_Toolkit, 2010, accessed: 2013-09-12.

[23] S. Holzer, R. 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 *Proc. IEEE/RSJ Int'l Conf. Intelligent Robots and Systems (IROS)*, Oct. 2012, pp. 2684–2689.

[24] K. Khoshelham and S. O. Elberink, "Accuracy and resolution of Kinect depth data for indoor mapping applications," *Sensors*, vol. 12, no. 2, pp. 1437–1454, 2012.

[25] J. Kopp, "Efficient numerical diagonalization of hermitian 3×3 matrices," *Int'l J. Modern Physics C*, vol. 19, no. 03, pp. 523–548, Mar. 2008.

[26] A. Hoover, G. Jean-Baptiste, X. Jiang, P. Flynn, H. Bunke, D. Goldgof, K. Bowyer, D. Eggert, A. Fitzgibbon, and R. Fisher, "An experimental comparison of range image segmentation algorithms," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 18, no. 7, pp. 673–689, July 1996.

[27] P. F. U. Gotardo, O. R. P. Bellon, and L. Silva, "Range image segmentation by surface extraction using an improved robust estimator," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, vol. 2, June 2003, pp. II–33–8.