2019 International Conference on Robotics and Automation (ICRA)
Palais des congres de Montreal, Montreal, Canada, May 20-24, 2019

Oriented Point Sampling for Plane Detection in Unorganized Point Clouds

Bo Sun[1] and Philippos Mordohai[1]

Abstract—
Plane detection in 3D point clouds is a crucial pre-processing step for applications such as point cloud segmentation, semantic mapping and SLAM. In contrast to many recent plane detection methods that are only applicable on organized point clouds, our work is targeted to unorganized point clouds that do not permit a 2D parametrization. We compare three methods for detecting planes in point clouds efficiently. One is a novel method proposed in this paper that generates plane hypotheses by sampling from a set of points with estimated normals. We named this method Oriented Point Sampling (OPS) to contrast with more conventional techniques that require the sampling of three unoriented points to generate plane hypotheses. We also implemented an efficient plane detection method based on local sampling of three unoriented points and compared it with OPS and the 3D-KHT algorithm, which is based on octrees, on the detection of planes on 10,000 point clouds from the SUN RGB-D dataset.

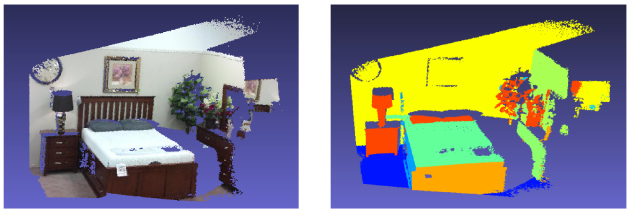
I. INTRODUCTION

As depth cameras and 3D sensors have become widely available recently, they are extensively used in various robotics and computer vision applications, such as Simultaneous Localization and Mapping (SLAM) [1], [2], [3], [4], dense 3D reconstruction [5], [6], [7], [8], [9], [10], [11] and 3D object recognition [12], [13]. 3D point clouds acquired by such depth cameras are generally noisy and redundant, and do not provide semantics of the scene.

An important first step to obtain more useful representations is to detect planar segments in the point clouds. In outdoor scenes, the ground is typically piecewise planar. In indoor scenes, most important surfaces, including ceilings, walls and floors, are planar. We can divide the planes into three groups: (1) horizontal planes, which are important due to their roles as support surfaces for other objects and as potentially traversable terrain for robots; (2) vertical planes, which are also important because many obstacles and the walls are vertical; (3) other planes, which constitute most of the other objects. It should be noted, however, that many of the recent plane detection approaches [14], [15], [16], [17], [17], [18] accept organized point clouds, i.e. $2\frac{1}{2}$-D depth images, as input. Here, we are interested in developing methods applicable on unorganized point clouds. This requires the search for neighboring points in 3D [19] since $2\frac{1}{2}$-D information cannot be exploited.

[1]Bo Sun and Philippos Mordohai are with the Department of Computer Science, Stevens Institute of Technology, Hoboken, NJ 07030, USA $\{bsun7, Philippos.Mordohai\}@stevens.edu$.

This work was supported in part by the National Institute Of Nursing Research of the National Institutes of Health under Award R01NR015371 and the National Science Foundation under Award IIS-1637761.



(a) Input point cloud (b) Output of OPS

Fig. 1. Sample input and output of OPS. (a) shows a point cloud from the SUN RGB-D dataset [20]. (b) shows the detected planes after merging.

In this paper, we seek to compare the effectiveness and efficiency of sampling oriented and unoriented points for plane detection in a RANSAC framework. Since sampling one oriented point is sufficient for generating a plane hypotheses, a small number of RANSAC iterations is required to detect the true planes. This, however, requires the computation of relatively accurate surface normals. Since we are interested in planes comprising many points, however, we only have to estimate normals for a small fraction of the points. On the other hand, a sample of three unoriented points can also lead to a plane equation requiring no pre-computations but more RANSAC iterations to draw three inliers from the same plane. This process can be accelerated by sampling the three points in a local neighborhood. The drawbacks are that it is harder to sample three inliers from the same plane, even locally, and that planes seeded this way are often noisy and may fail the hypothesis verification step.

To enable this comparison, we propose a novel algorithm called Oriented Points Sampling (OPS), which is based on sparsely sampling points from the point cloud, estimating their normals, using these oriented points to generate plane hypotheses and finally verifying the hypotheses on all points. The second algorithm is a modification of the work of Biswas and Veloso [15], [21], which is referred to as Fast Sampling Point Filtering (FSPF). FSPF can efficiently detect planes in depth images by sampling points to build and verify plane hypotheses in the vicinity of an original point sampled uniformly from the depth image. We modify the algorithm to sample in spheres around the original point. We evaluate the effectiveness and the computational efficiency of both methods, as well as 3D-KHT algorithm [22], on the SUN RGB-D dataset [20] that comprises over 10,000 point clouds. Figure 1 shows an example from the SUN RGB-D dataset.

The main contributions of this paper are the following:

- We propose OPS, a fast, RANSAC-based, plane detection method in unorganized point clouds that requires a minimal sample of only one oriented point to generate

a hypothesis.
- We compare OPS on classifying the points of a point cloud according to plane orientation and on plane segmentation with two allternative approaches: an extension of FSPF that samples three unoriented point proposed here and 3D-KHT that operates in octrees.

II. RELATED WORK

There are many different methods of plane detection. These methods can be generally classified into three categories: (i) point clustering; (ii) region growing; (iii) RANSAC-based plane fitting.

Methods based on point clustering rely on similarity measures, such as proximity between points and angle between surface normals. Strom et al. [23] generated an 8-connected graph from successive laser scans, colored using RGB images, and clustered neighboring points according to differences in color and angle between surface normals. Holz et al. [24] computed per-point normals and clustered the points according to normal orientation and offset of hypothetical planes going through the oriented points. Enjarini and Graser [25] computed the gradient of depth feature on the depth image and performed a voting-based clustering process, followed by region merging. Pham et al. [26] initially cluster the input into supervoxels and then perform clustering on the adjacency graph of the supervoxels. Adjacent regions are then classified as belonging to the same or different objects. Lee and Campbell [27] proposed an efficient surface normal estimation method which discretized the space into grid cells with occupancy information and infered the surface normal direction for each cell utilizing the occupancy information of the neighboring cells. Planes are detected by clustering points with similar normals. Dzitsiuk et al. [11] presented a fast and robust plane detection method for de-noising, 3D reconstruction and hole filling based on Signed Distance Functions (SDFs). The scene is discretized into a set of volumes which contain blocks of voxels with SDF values. Two ways to generate plane candidates in each volume are compared: RANSAC and least squares fitting on the SDF.

3D-KHT [22] is based on a Hough transform and works by clustering approximately co-planar points and by casting votes for these clusters on a spherical accumulator using a trivariate Gaussian kernel. An octree is used to recursively subdivide the points while a coplanarity test is performed in each node. The recursive subdivision process stops when the points in the node pass the coplanarity test or when there are not enough points in the node.

Methods based on region growing select seed points or regions as the original patches and cluster compatible points with each patch. Poppinga et al. [14] proposed a plane fitting algorithm based on region growing followed by a polygonalization algorithm to find a set of convex polygons covering the same area as the set of triangles that is produced by connecting all triplets of points neighboring in the range image. Deschaud and Goulette [28] presented a fast and accurate algorithm to detect planes in unorganized point clouds using filtered normals and voxel growing. Holz and Behnke [29] constructed a triangular mesh from the input point cloud and computed local surface normals and curvature estimates. The resulting information was used to segment the range images into planar regions and other geometric primitives. Arbeiter et al. [30] proposed an efficient normal based region growing method to obtain segments containing points with common geometric properties. Planes are produced after classifying the segments according to point feature descriptors. Feng et al. [17] performed agglomerative hierarchical clustering on the point graph to merge nodes belonging to the same plane. The extracted planes were refined using pixel-wise region growing. Monszpart et al. [31] proposed a global approach to simultaneously detect a set of planes along with their relations. Zermas et al. [32] extracted a set of seed points with low height values which were then used to estimate the initialize the ground surface. Then, points near the initial ground plane were used as seeds to refine the estimate of the ground plane in an iterative process.

RANSAC-based methods address plane detection by sampling points from the point cloud, fitting planar models to them and accepting hypotheses that have accumulated sufficient support. Oehler et al. [33] extracted surface elements at multiple resolutions, from coarse to fine. Surface elements that cannot be associated with planes from coarser resolutions are grouped into coplanar clusters using a Hough transform. Connected components are extracted from these clusters and planes are fit using RANSAC. Gallo et al. [34] proposed a modification of the RANSAC algorithm, dubbed CC-RANSAC, that only considers the largest connected components of inliers to evaluate the fitness of a candidate plane. Hulik et al. [35] split the depth map into square tiles and applied RANSAC within each tile. Then, two seed-fill algorithms are successively applied to group all connected planes in the current tile and across tile borders. Qian and Ye [36] proposed NCC-RANSAC, which performs a normal coherence check on all points of the inlier patches and removes points whose normal directions are contradictory to that of the fitted plane. The removed points are used to form candidate planes, which are recursively clustered. Marriott et al. [18] extract planar models from depth-data by fitting the data with a piecewise-linear Gaussian mixture regression model and fusing contiguous and coplanar components to generate the final set of planes.

An integral part of our paper is the FSPF algorithm of Biswas and Veloso [15], [21], which groups points in the depth image and classifies them as belonging to planes or not. To form a plane hypothesis, three points are sampled in a local area, while support for the hypothesis is also estimated locally for efficiency. The detected planes are converted into a set of convex polygons, which are merged across successive depth images. See Section III for more details.

III. APPROACH

In this section, we present a method to detect planes of all orientations in a point cloud. We show results on horizontal planes, vertical planes and other planes which are of interest as support surfaces, bounding walls and other

objects. We introduce two methods. The first one is Oriented Point Sampling (OPS) which can find all-direction planes in real time. The second one is a fast sampling plane filtering method inspired by [15], [21] which we use to compare with OPS in accuracy and speed.

A. Oriented Point Sampling

OPS accepts as input a point cloud denoted by $P = \{p_1, p_2, p_3, \cdots, p_N\}, p_i \in \mathbb{R}^3$, where N is the number of points. The output is a set of planes Π. Each plane is stored in a 2×3 matrix M. The first row of M is the plane's centroid and the second row of M is its normal.

Processing begins by estimating the normals of a fraction α_s of the points. On one hand, the availability of oriented points allows the generation of plane hypotheses by sampling just one inlier. On the other hand, accurate normal estimation is computationally expensive and the planes of interest should comprise large numbers of points. Therefore, we only compute the normals of a subset of the points, but use all points for hypotheses verification.

Normal Estimation: Given the sampling rate α_s, we uniformly sample a subset of the points $P_s = \left\{p_1^{(s)}, p_2^{(s)}, p_3^{(s)}, \cdots, p_{N_s}^{(s)}\right\}, p_i^{(s)} \in \mathbb{R}^3$, where N_s is the cardinality of P_s. We assume that surface normals can be estimated by performing SVD on the neighbors of the reference point, for which we want to estimate the normal.

The current reference point is denoted by $p_i = (p_{i,x}, p_{i,y}, p_{i,z})$ and its normal is denoted by $n_i = (n_{i,x}, n_{i,y}, n_{i,z})$. The k nearest neighbors of p_i are found using a k-d tree and denoted by $Q_i = \{q_{i1}, q_{i2}, q_{i3}, \cdots, q_{ik}\}, q_{ij} \in P$ and $q_{ij} \neq p_i$. k is a key parameter for the accuracy of normal estimation. According to [37], we compute the following matrix M_i of each reference point and compute the normal as the eigenvector corresponding to the minimum eigenvalue of M_i.

$$M_i = \sum_{q_{ij} \in Q_i} e^{-\frac{\|q_{ij}-p_i\|_2^2}{2\sigma^2}} \frac{(q_{ij}-p_i)(q_{ij}-p_i)^T}{\|q_{ij}-p_i\|_2^2} \quad (1)$$

This formula includes a weight term $e^{-\frac{\|q_{ij}-p_i\|_2^2}{2\sigma^2}}$ in order to reduce the effects of neighboring points which are far from the reference point p_i. All vectors connecting neighbors to the reference point are normalized.

One-Point RANSAC: After normal computation, we apply one-point RANSAC to find the largest plane. As opposed to conventional RANSAC-based plane detection, that requires minimal samples of three points, OPS requires only one point with its normal to define a plane. Given the set of sampled points P_s, we randomly pick one point with its normal which determine a plane. We compute the distance from all the other sampled points to this plane and count the number of inliers that are within a distance threshold θ_h. We also define a threshold value θ_N to decide the minimum number of points for a plane to be accepted.

We iterate the above steps to find the plane model with the largest number of inliers. The number of iterations N_{iter} is adaptively determined by $N_{iter} = \frac{log(1-p)}{log(1-(1-e))}$ [38]. N_{iter} is initialized as the number of all the points in the point cloud. p is the probability that at least one random sample is free from outliers and e is the proportion of outliers among all points. e is updated in each iteration as we find more inliers. Since we only need to select one true inlier, the number of iterations is much smaller than alternatives that require sampling three inliers. After we find the largest plane, we re-estimate the normal using SVD on all inliers. OPS is summarized in Algorithm 1.

Algorithm 1: OPS

Input : An unorganized point cloud $P = \{p_1, p_2, p_3, \cdots, p_N\}, p_i \in \mathbb{R}^3$, sampling rate α_s, k, p, distance threshold θ_h, inlier threshold θ_N

Output: The largest horizontal plane model M, the inlier set $bestInliers$

Randomly sample $\alpha_s N$ 3D points $P_s = \left\{p_1^{(s)}, p_2^{(s)}, p_3^{(s)}, \cdots, p_{N_s}^{(s)}\right\}, p_i^{(s)} \in \mathbb{R}^3$
Compute their normals using local SVD on k neighbors
The normals of the sampled points are $\left\{n_1^{(s)}, n_2^{(s)}, n_3^{(s)}, \cdots, n_{N_s}^{(s)}\right\}$
$iter \leftarrow 0$
$inNum \leftarrow 0$
$N_{iter} \leftarrow N$
while $iter < N_{iter}$ **do**
  $inliers \leftarrow NULL$
  $randInd \leftarrow rand()\%N_s$
  for *each* $p_i^{(s)} \in P_s$ **do**
    $p_r \leftarrow p_i^{(s)} - p_{randInd}^{(s)}$
    $d \leftarrow p_r \cdot N_{randInd}^{(s)}$
    if $d < \theta_h$ **then**
      Add $p_i^{(s)}$ to $inliers$
    end
  end
  if $inliers.size() > \theta_N$ **then**
    if $inliers.size() > inNum$ **then**
      $inNum \leftarrow inliers.size()$
      $bestInliers \leftarrow inliers$
      $e \leftarrow 1 - \left(\frac{inliers.size()}{N_h}\right)$
      $N_{iter} \leftarrow \frac{log(1-p)}{log(1-(1-e))}$
    end
  end
  $iter \leftarrow iter + 1$
end
for *each* $bestInliers_i \in bestInliers$ **do**
  $pointsInlier(i,:) \leftarrow bestInliers_i$
end
$(u, v) \leftarrow SVDCompute(pointsInlier)$
$M(1,:) \leftarrow v(2,:)$
$M(0,:) \leftarrow$ centroid of plane
return M, $bestInliers$

Detection of Multiple Planes: To detect all planes, we apply OPS multiple times. After a plane has been detected, we remove all its inliers from the point cloud and apply OPS on the remaining points until there are not enough points (θ_N) to constitute a plane.

Detecting planes in multiple orientations: To extract basic scene semantics, we divide the detected planes into three groups: vertical, horizontal and other. We have tried two strategies to detect these three groups of planes after sampling some points. (1) Apply OPS on the sampled points and divide the detected planes into three groups. (2) Divide the sampled points into three groups based on the estimated normals, and then apply OPS separately on each of the three groups. We chose the second strategy due to its superior performance in our experiments, as shown in Section IV.

B. Fast Sampling Plane Filtering

In order to compare with OPS, we make some modifications to the Fast Sampling Plane Filtering method [15], [21], which was designed for depth images. Since we are interested in unorganized point clouds, which, unlike RGB-D images, do not take 2D parametrizations, we make the following two modifications to FSPF. First, while the original FSPF samples neighboring points by adding random integers to the image coordinates of the reference point, we sample neighboring points in a sphere around the reference point. Second, we design a new plane merging method described in Section III-C. We use FSPF to refer to the algorithm including these modifications in this paper.

FSPF (Algorithm 2) begins by sampling three points $p_{r,0}, p_{r,1}, p_{r,2}$ from the 3D point cloud. The first point $p_{r,0}$ is selected uniformly in the point cloud, while $p_{r,1}$ and $p_{r,2}$ are sampled within a sphere with a radius r_1 around $p_{r,0}$. We use a k-d tree to find the neighbors in the sphere. $p_{r,0}$, $p_{r,1}$ and $p_{r,2}$ determine a plane π and we use a cross product to compute its normal. A search sphere with radius r_2 is then constructed around point $p_{r,0}$ using the k-d tree and an additional N_{loc} local samples are randomly sampled in the search sphere. The plane fitting error is computed as the distance between each point in the local samples and the plane π. Points are considered inliers if their error is less than θ_h, which has the same value as in OPS. If more than $\alpha_{min}N_{loc}$ points in the search sphere are classified as inliers, then all the points in the sphere are classified as inliers of π and π is recorded as one of the planes found by FSPF. We repeat the above steps for K_{max} iterations or until we find N_{max} inlier points.

We use FSPF to find all the planes in a point cloud and divide the detected planes into three groups: vertical, horizontal and other. Filtering planes after the initial three points are sampled is ineffective because the normals estimated via the cross product are very noisy. Grouping these planes into vertical, horizontal and other at this stage leads to the detection of numerous planes that are not correctly classified once their normals are re-estimated using all inliers. Filtering planes only after re-estimation is by far the best strategy.

Algorithm 2: Fast Sampling Plane Filtering

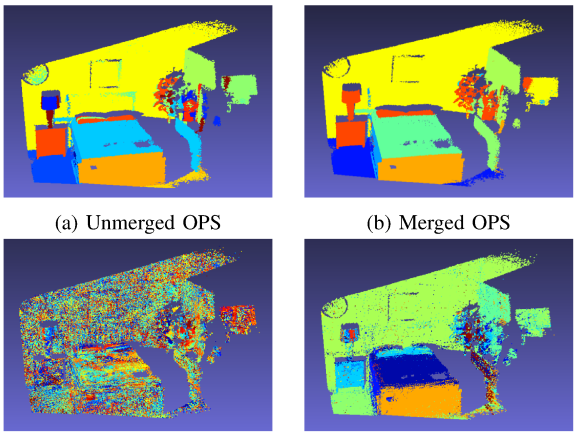
Input : An unorganized point cloud $P = \{p_1, p_2, p_3, \cdots, p_N\}, p_i \in \mathbb{R}^3$, N_{max}, K_{max}, N_{loc}, α_{min}, θ_h, r_1, r_2

Output: A set of planes Π and the corresponding inliers

$\Pi \leftarrow NULL$
$n, k \leftarrow 0$
while $n < N_{max} \bigwedge k < K_{max}$ **do**
  $k \leftarrow k + 1$
  $i = rand(1, N)$
  $p_{r,0} \leftarrow p_i$
  $P_1 \leftarrow kdtree.radiusSearch(p_{r,0}, r_1)$
  $p_{r,1} \leftarrow RandomlyPickPoint(P_1)$
  $p_{r,2} \leftarrow RandomlyPickPoint(P_1)$
  $n_\pi = \frac{(p_{r,1}-p_{r,0})\times(p_{r,2}-p_{r,0})}{\|(p_{r,1}-p_{r,0})\times(p_{r,2}-p_{r,0})\|_2}$
  $\hat{\Pi} \leftarrow NULL$
  $N_{inlier} \leftarrow 0$
  $P_2 \leftarrow kdtree.radiusSearch(p_{r,0}, r_2)$
  for $j \leftarrow 3 \cdots N_{loc}$ **do**
    $p_j \leftarrow RandomlyPickPoint(P_2)$
    $e = abs(n_\pi \cdot (p_j - p_{r,0}))$
    if $e < \theta_h$ **then**
      Add p_j to $\hat{\Pi}$
      $N_{inlier} \leftarrow N_{inlier} + 1$
    end
  end
  if $N_{inlier} > \alpha_{min} N_{loc}$ **then**
    for *each* $\hat{\Pi}_i \in \hat{\Pi}$ **do**
      $inlierPoints(i, :) \leftarrow \hat{\Pi}_i$
    end
    $(u, v) \leftarrow SVDCompute(inlierPoints)$
    $M(1, :) \leftarrow v(2, :)$
    $M(0, :) \leftarrow$ centroid of plane
    Add M to Π
    Add inlierPoints to $allInliers$ $n \leftarrow n + N_{inlier}$
  end
end
return Π, $allInliers$

C. Merging Process for Both Methods

FSPF returns a lot of small planes because it searches for planes locally. The results of OPS also require some merging, even though it produces larger planes in general. We merge small planes by applying a coplanarity test on each pair of planes. Specifically, we test whether the (1) angle between the normals of the two planes, (2) the distance between the centroid of the second plane and the first plane, and (3) the distance between the centroid of the first plane and the second plane are below appropriate thresholds. If a pair of planes passes the coplanarity test, we merge them into one plane and re-estimate the normal of the new plane using SVD on all points. To ensure a fair comparison, we apply the same merging process to both FSPF and OPS. Some results of merging are shown in Figure 2.



(a) Unmerged OPS (b) Merged OPS

(c) Unmerged FSPF (d) Merged FSPF

Fig. 2. The first row is the output of OPS before and after merging. The second row is the output of FSPF before and after merging.

IV. EXPERIMENTS

We perform experiments on the SUN RGB-D dataset [20] to compare the performance and run time of OPS, FSPF, modified according to Section III and 3D-KHT [22]. Using the ground truth planes (see below), we compute the average classification and segmentation accuracy on 10,355 point clouds for all three methods. Classification accuracy refers to assigning each point to the correct plane orientation, while segmentation accuracy refers to assigning each point to the correct plane. We use the Hungarian Algorithm to solve the assignment problem for segmentation accuracy.

A. Datasets and Experimental Setup

In the experiments, we use the SUN RGB-D dataset [20] which is captured by four different sensors and contains 10,335 RGB-D images, which *we treat as unorganized point clouds*. The entire dataset is densely annotated and includes 146,617 2D polygons and 64,593 3D bounding boxes. We convert the RGB-D images to point clouds, which are the only inputs to the algorithms. To detect the ground truth planes, we estimate the normals of all the points and iteratively use region growing considering distance and normal similarity to extract dense planes. We use 0.05 m as the distance threshold θ_h and we use 7 degrees for the threshold of angle between normals. Points on small planes, with under 50 points, are labeled as "other". Points on horizontal planes are labeled horizontal and points on vertical planes are labeled vertical.

We run our version of FPSF using the following parameters: the number of local samples N_{loc} is 80, the plane offset error θ_h is 0.05 m, and the minimum inlier fraction α_{min} to accept a local sample is 0.8. We vary r_1, the radius of the sphere for finding $p_{r,1}$ and $p_{r,2}$, and r_2, the radius of the search sphere for finding inliers. For 3D-KHT, the minimum number of samples required in a cluster is 30 and the octree level for checking for approximate coplanarity is 5. For OPS, the tolerance to classify a surface normal as vertical or horizontal is 7 degrees, the probability p for adaptively determining RANSAC iterations is 0.99, and the minimum number of points for plane fitting θ_N is 20. We vary the fraction of points we sample and the number of nearest neighbors for normal estimation. Timing results are reported for single-threaded C++ implementations on an Intel Core i7 7700 processor.

B. Results

First, we try different values of r_1 and r_2 for FSPF. We use two configurations: (1) $r_2 = r_1$; (2) $r_2 = 2r_1$. The performance metrics and run time of FSPF are shown in Table I. The performance improves as r_1 increases. FSPF performs the best when r_1 is 0.1 m and $r_2 = 2r_1$.

TABLE I
CLASSIFICATION ACCURACY AND RUN TIME OF FSPF.

r_1	r_2	Accuracy	FSPF Time	Merge Time	Total Time
0.03	0.03	61.34%	0.0974 sec	0.1895 sec	0.2997 sec
0.05	0.05	68.07%	0.1102 sec	0.2103 sec	0.3003 sec
0.07	0.07	69.68%	0.2106 sec	0.1937 sec	0.4043 sec
0.1	0.1	71.14%	0.2479 sec	0.2145 sec	0.4642 sec
0.3	0.3	73.12%	2.6423 sec	0.3293 sec	2.9716 sec
0.03	0.06	69.01%	0.1139 sec	0.2415 sec	0.3554 sec
0.05	0.1	71.11%	0.1973 sec	0.2487 sec	0.4460 sec
0.07	**0.14**	**72.50%**	**0.3441 sec**	**0.2785 sec**	**0.6226 sec**
0.1	0.2	73.61%	0.7277 sec	0.3180 sec	1.0457 sec
0.3	0.6	66.26%	2.7624 sec	0.4960 sec	3.2584 sec

The run time of FSPF consists of plane detection and merging. The former ranges from 0.1 to 0.7 seconds. FSPF plane detection time becomes longer as r_1 increases. When r_1 becomes larger than 0.3, the detection is very slow. Since our focus is on real-time plane detection, we only show results for $r_1 \leq 0.3m$. The merging time ranges from 0.2 to 0.5 seconds. FSPF detects more planes when r_1 and r_2 become larger. Merging takes longer when r_1 increases, since there are more planes to be merged.

Second, we evaluate the accuracy and run time of 3D-KHT, as shown in Table II. 3D-KHT uses a spherical accumulator to cast votes based on Hough transform. We tried different number of cells in the angular ϕ and radius r direction. The more cells it uses, the more time it costs and the more accurate planes it detects.

TABLE II
CLASSIFICATION ACCURACY AND RUN TIME OF 3D-KHT.

Angular bins	Radial bins	Accuracy	Run Time
30	300	67.08%	0.0837 sec
40	400	67.86%	0.1448 sec
60	600	69.53%	0.3674 sec
80	**800**	**71.01%**	**0.7941 sec**

Last, we evaluate the effects of different sampling rates and different numbers of nearest neighbors on the performance of OPS. The experimental results are shown in Table III. The average classification accuracy of OPS improves as the sampling rate increases. Using a larger number of nearest neighbors for normal estimation improves the results of OPS at the cost of longer run time. However, when the sampling rate is more than 0.5% even 10 nearest neighbors are

TABLE III

CLASSIFICATION ACCURACY AND RUN TIME OF OPS. THE COLUMNS SHOW THE SAMPLING RATE, THE NUMBER OF NEAREST NEIGHBORS, AVERAGE ACCURACY, RUN TIME FOR PLANE DETECTION AND TOTAL TIME, INCLUDING MERGING.

Sampl. rate	NN	Accuracy	OPS Time	Total Time
0.3%	10	71.52%	0.0244 sec	0.0359 sec
0.3%	20	72.18%	0.0347 sec	0.0460 sec
0.3%	30	72.29%	0.0458 sec	0.0565 sec
0.5%	10	77.11%	0.0452 sec	0.0625 sec
0.5%	20	77.48%	0.0644 sec	0.0803 sec
0.5%	30	77.59%	0.0833 sec	0.0987 sec
0.7%	10	80.04%	0.0656 sec	0.0853 sec
0.7%	20	80.28%	0.0906 sec	0.1093 sec
0.7%	30	80.34%	0.1184 sec	0.1363 sec
1%	10	82.55%	0.1008 sec	0.1233 sec
1%	20	82.66%	0.1375 sec	0.1587 sec
1%	30	82.71%	0.1759 sec	0.1967 sec
3%	10	86.91%	0.4092 sec	0.4377 sec
3%	20	87.23%	0.5096 sec	0.5366 sec
3%	**30**	**87.39%**	**0.6188 sec**	**0.6455 sec**

sufficient for OPS to achieve high accuracy. As mentioned in Section III, we tried the first strategy for detecting planes in multiple orientations. The average accuracy is 74.25%, 75.04% and 75.89% when we estimate normals for 3% of the points and the number of nearest neighbors is 10, 20 and 30. This is worse than the last three rows of Table III.

OPS is better than FSPF with any r_1 in classification accuracy when the sampling rate is between 0.5% and 3%. When $r_1 = 0.1, r_2 = 2r_1$, FSPF achieves the highest accuracy with a run time of about 1 second. In addition to higher accuracy, OPS is also faster than FSPF when the sampling rate is between 0.5% and 3%. The merging process of OPS is also much faster than that of FSPF because OPS detects larger planes than FSPF. On average, in each point cloud OPS detects 34.62 planes before merging and 27.59 planes after merging. The corresponding numbers for FSPF are 386.92 and 29.44. The ground truth contains on average 25.38 planes. The left column of Fig. 3 shows the results of classifying the points according to the three possible orientations and the right column shows results where each plane is marked in a pseudo-random color.

We pick the best parameter settings of the three methods and evaluate segmentation accuracy, as shown in Table IV. The best parameter setttings are: (1) $r_1 = 0.07, r_2 = 0.14$ for FSPF; (2) 80 angular bins and 800 radius bins for 3D-KHT; (3) 3% sampling rate and 30 nearest neighbors for OPS. OPS is able to assign points to the correct plane with much higher accuracy. Note that the ground truth was generated by region growing which different than all three algorithms.

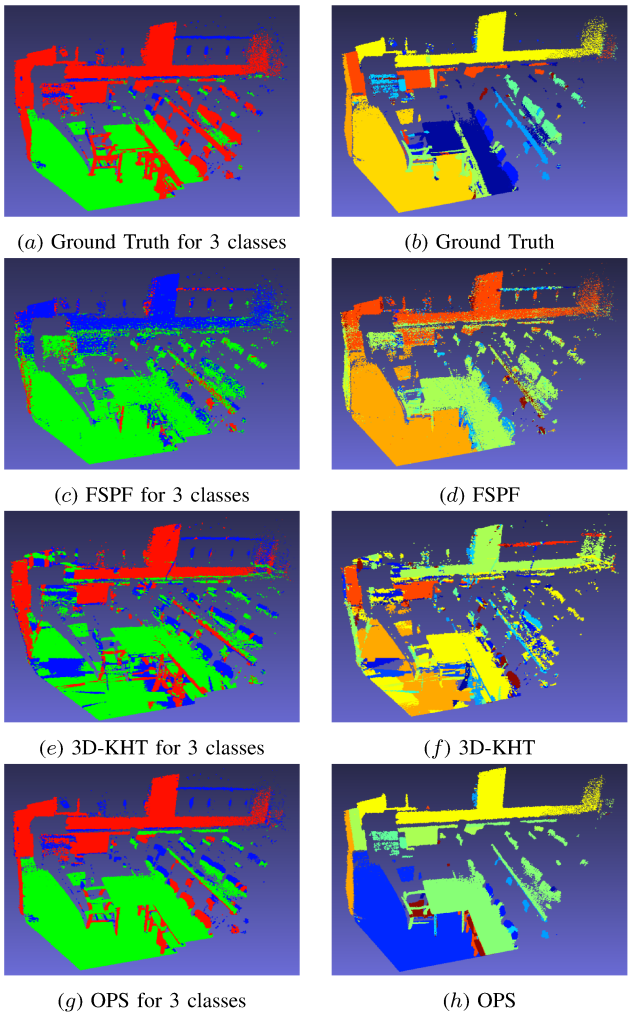
TABLE IV

SEGMENTATION ACCURACY OF FSPF, 3D-KHT AND OPS.

FSPF	3D-KHT	OPS
74.72%	62.26%	82.99%

V. CONCLUSIONS

In this paper, we introduced a new sampling-based method for detecting plane in point clouds and compared it with two other approaches. We introduced OPS which is based on sparsely sampling points from the point cloud, estimating their normals and then using these oriented points to generate plane hypotheses for verification. We modified FSPF, which was originally designed for depth images, to be applicable to unorganized point clouds. We also used the 3D-KHT algorithm which relies on an octree to cluster approximately coplanar points. We experimentally evaluated the accuracy and computational efficiency of all methods on the large scale SUN RGB-D dataset. Our conclusions are that OPS is superior in both speed and accuracy. These experiments provide evidence that may help resolve the debate on whether investing computational resources for normal estimation is beneficial for plane detection. While estimating the normals of all points using more than 10 nearest neighbors and a large number of queries to a k-d tree is too costly in terms of computation, it is sufficient to compute normals for only a small fraction of the points, as low as 0.3% in these experiments. The availability of these privileged, oriented points allows the use of a one-point RANSAC scheme which can detect almost all planes in the scene.



(*a*) Ground Truth for 3 classes (*b*) Ground Truth

(*c*) FSPF for 3 classes (*d*) FSPF

(*e*) 3D-KHT for 3 classes (*f*) 3D-KHT

(*g*) OPS for 3 classes (*h*) OPS

Fig. 3. Some results of FSPF, 3D-KHT and OPS. The left and right columns show classification and segmentation results, respectively.

REFERENCES

[1] C. Kerl, J. Sturm, and D. Cremers, "Dense visual SLAM for RGB-D cameras," in *IROS*, 2013, pp. 2100–2106.

[2] T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard, and J. McDonald, "Robust real-time visual odometry for dense rgb-d mapping," in *ICRA*, 2013, pp. 5724–5731.

[3] L. Ma, C. Kerl, J. Stückler, and D. Cremers, "CPA-SLAM: Consistent plane-model alignment for direct RGB-D SLAM," in *ICRA*, 2016, pp. 1285–1291.

[4] S. Yang, Y. Song, M. Kaess, and S. Scherer, "Pop-up SLAM: semantic monocular plane SLAM for low-texture environments," in *IROS*, 2016, pp. 1222–1229.

[5] A. Nüchter and J. Hertzberg, "Towards semantic maps for mobile robots," *Robotics and Autonomous Systems*, vol. 56, no. 11, pp. 915–926, 2008.

[6] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011, pp. 127–136.

[7] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "RGB-D mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 647–663, 2012.

[8] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3d reconstruction at scale using voxel hashing," *ACM Trans. on Graphics*, vol. 32, no. 6, p. 169, 2013.

[9] S. Choi, Q.-Y. Zhou, and V. Koltun, "Robust reconstruction of indoor scenes," in *IEEE. Conf. on Computer Vision and Pattern Recognition*, 2015, pp. 5556–5565.

[10] M. Klingensmith, I. Dryanovski, S. Srinivasa, and J. Xiao, "Chisel: Real time large scale 3d reconstruction onboard a mobile device using spatially hashed signed distance fields." in *Robotics: Science and Systems*, 2015.

[11] M. Dzitsiuk, J. Sturm, R. Maier, L. Ma, and D. Cremers, "De-noising, stabilizing and completing 3d reconstructions on-the-go using plane priors," in *ICRA*, 2017, pp. 3976–3983.

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

[13] X. Qian and C. Ye, "3d object recognition by geometric context and gaussian-mixture-model-based plane classification," in *ICRA*, 2014, pp. 3910–3915.

[14] J. Poppinga, N. Vaskevicius, A. Birk, and K. Pathak, "Fast plane detection and polygonalization in noisy 3d range images," in *IROS*, 2008, pp. 3378–3383.

[15] J. Biswas and M. Veloso, "Fast sampling plane filtering, polygon construction and merging from depth images," in *RSS, RGB-D Workshop*, 2011.

[16] ——, "Depth camera based indoor mobile robot localization and navigation," in *ICRA*, 2012, pp. 1697–1702.

[17] C. Feng, Y. Taguchi, and V. R. Kamat, "Fast plane extraction in organized point clouds using agglomerative hierarchical clustering," in *ICRA*, 2014, pp. 6218–6225.

[18] R. T. Marriott, A. Paschevich, and R. Horaud, "Plane-extraction from depth-data using a gaussian mixture regression model," *arXiv preprint arXiv:1710.01925*, 2017.

[19] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 36, no. 11, pp. 2227–2240, 2014.

[20] S. Song, S. P. Lichtenberg, and J. Xiao, "SUN RGB-D: A rgb-d scene understanding benchmark suite," in *IEEE. Conf. on Computer Vision and Pattern Recognition*, 2015, pp. 567–576.

[21] J. Biswas and M. Veloso, "Planar polygon extraction and merging from depth images," in *IROS*, 2012, pp. 3859–3864.

[22] F. A. Limberger and M. M. Oliveira, "Real-time detection of planar regions in unorganized point clouds," *Pattern Recognition*, vol. 48, no. 6, pp. 2043–2053, 2015.

[23] J. Strom, A. Richardson, and E. Olson, "Graph-based segmentation for colored 3D laser point clouds," in *IROS*, 2010, pp. 2131–2136.

[24] D. Holz, S. Holzer, R. B. Rusu, and S. Behnke, "Real-time plane segmentation using RGB-D cameras," in *Robot Soccer World Cup*, 2011, pp. 306–317.

[25] B. Enjarini and A. Gräser, "Planar segmentation from depth images using gradient of depth feature," in *IROS*, 2012, pp. 4668–4674.

[26] T. T. Pham, M. Eich, I. Reid, and G. Wyeth, "Geometrically consistent plane extraction for dense indoor 3D maps segmentation," in *IROS*, 2016, pp. 4199–4204.

[27] D. J. Lee and M. E. Campbell, "An efficient probabilistic surface normal estimator," in *ICRA*, 2016, pp. 2248–2254.

[28] 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 *"Int. Symposium on 3D Data Processing, Visualization and Transmission"*, 2010.

[29] D. Holz and S. Behnke, "Fast range image segmentation and smoothing using approximate surface reconstruction and region growing," *Intelligent Autonomous Systems*, pp. 61–73, 2013.

[30] G. Arbeiter, S. Fuchs, J. Hampp, and R. Bormann, "Efficient segmentation and surface classification of range images," in *ICRA*, 2014, pp. 5502–5509.

[31] A. Monszpart, N. Mellado, G. J. Brostow, and N. J. Mitra, "RAPter: rebuilding man-made scenes with regular arrangements of planes." *ACM Trans. on Graphics*, vol. 34, no. 4, pp. 103–1, 2015.

[32] D. Zermas, I. Izzat, and N. Papanikolopoulos, "Fast segmentation of 3d point clouds: A paradigm on lidar data for autonomous vehicle applications," in *ICRA*, 2017, pp. 5067–5073.

[33] B. Oehler, J. Stueckler, J. Welle, D. Schulz, and S. Behnke, "Efficient multi-resolution plane segmentation of 3D point clouds," *Intelligent Robotics and Applications*, pp. 145–156, 2011.

[34] O. Gallo, R. Manduchi, and A. Rafii, "CC-RANSAC: Fitting planes in the presence of multiple surfaces in range data," *Pattern Recognition Letters*, vol. 32, no. 3, pp. 403–410, 2011.

[35] R. Hulik, V. Beran, M. Spanel, P. Krsek, and P. Smrz, "Fast and accurate plane segmentation in depth maps for indoor scenes," in *IROS*, 2012, pp. 1665–1670.

[36] X. Qian and C. Ye, "NCC-RANSAC: a fast plane extraction method for 3-D range data segmentation," *IEEE Transactions on Cybernetics*, vol. 44, no. 12, pp. 2771–2783, 2014.

[37] K. Jordan and P. Mordohai, "A quantitative evaluation of surface normal estimation in point clouds," in *IROS*, 2014, pp. 4220–4226.

[38] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, 2004.