# A Robust Statistics Approach for Plane Detection in Unorganized Point Clouds

Abner M. C. Araújo, Manuel M. Oliveira*

*Universidade Federal do Rio Grande do Sul*
*Instituto de Informática - PPGC - CP 15064*
*91501-970 - Porto Alegre - RS - BRAZIL*

**Abstract**

Plane detection is a key component for many applications, such as industrial reverse engineering and self-driving cars. However, existing plane-detection techniques are sensitive to noise and to user-defined parameters. We introduce a fast deterministic technique for plane detection in unorganized point clouds that is robust to noise and virtually independent of parameter tuning. It is based on a novel planarity test drawn from robust statistics and on a split and merge strategy. Its parameter values are automatically adjusted to fit the local distribution of samples in the input dataset, thus lending to good reconstruction of even small planar regions. We demonstrate the effectiveness of our solution on several real datasets, comparing its performance to state-of-art plane detection techniques, and showing that it achieves better accuracy, while still being one of the fastest.

*Keywords:* Plane detection, Region growing, Robust statistics, Unorganized point clouds.

*Corresponding author. Tel.: +55 51 3308 6821; Fax: +55 51 3308 7308.
*Email addresses:* amcaraujo@inf.ufrgs.br (Abner M. C. Araújo), oliveira@inf.ufrgs.br (Manuel M. Oliveira)
*URL:* http://inf.ufrgs.br/~amcaraujo (Abner M. C. Araújo), http://inf.ufrgs.br/~oliveira (Manuel M. Oliveira)

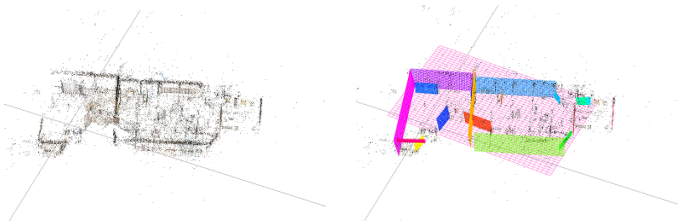

Figure 1: Example of plane detection using our technique. (left) Input point cloud. (right) Detected planar patches. Notice how it is able to extract delimited planes from connected regions, despite the high level of noise in this point cloud.

## 1. Introduction

Detecting planes in unorganized point clouds is key to several emerging applications, such as self-driving cars [1], robotic navigation [2], and forensics [3]. It is also essential to more conventional applications, such as reverse engineering [4, 5, 6, 7, 8, 3], camera calibration [9], object recognition [10, 11], object reconstruction [12], denoising [13], augmented reality [14, 15], and segmentation [16, 17]. Since point clouds are inherently noisy and datasets containing several millions of samples are now commonplace, detecting planar regions in point clouds requires both robustness to noise and performance scalability. For applications such as reverse engineering, accurate reconstruction is also a crucial aspect. Unfortunately, most available techniques for detecting planar structures in point clouds are either sensitive to noise or computationally expensive. Moreover, most previous techniques use parameters that need to be tunned for different datasets, which is undesirable and time consuming.

We present an efficient $O(n \log n)$ deterministic technique for plane detection in unorganized point clouds. *Our solution uses robust statistics to derive a novel planarity test that is mostly insensitive to outliers.* More specifically, *we use robust measures of distance to plane and normal deviation to detect and remove outliers, as well as to automatically adjust our method's parameters to the local distribution of samples in the input dataset.* This results in a robust-to-noise approach, which is also virtually independent of parameter tuning, and handles point clouds of large sizes and variable sampling distributions.

Our technique uses a subdivision strategy to first detect small planar regions in the point cloud. This is done to improve the accuracy of the overall detection process (Section 3.1). Since it is undesirable to have a plane fragmented into multiple small patches, it subsequently employs an iterative procedure to grow and merge such patches (Section 3.3). This results in a fast and accurate solution capable of automatically delimiting the detected planar regions. The growth procedure is similar to most region-growing techniques, but unlike previous solutions, our technique does not require user-specified parameters.

We demonstrated the effectiveness of our solution by performing a detailed comparison with the most popular, as well as with the most recent approaches for plane detection. The comparisons were carried out on seven datasets chosen to cover various real point-cloud characteristics, such as variable sampling density, noise level, number of samples, number of planes, and different acquisition sensors. Our technique performed well on all datasets, achieving the best accuracy measured in terms of average precision, recall, and F1-score over all datasets, while still being one of the fastest.

Figure 1 illustrates the use of our technique to automatically extract delimited planar regions from a noisy point cloud. The image on the left shows the input dataset, while the one on the right shows the detected planes. Note how our technique is able to detect and delimit these regions despite the high level of noise in the point cloud.

The **contributions** of this work include:

- *A technique for detecting planes in unorganized point clouds that is robust to noise and virtually independent of parameter tunning* (Section 3). Our solution is more accurate than previous techniques, while being computationally efficient;

- *A novel planarity test based on robust statistics* that is less dependent on the noise scale of the point cloud than existing solutions (Section 3.2). It allows our technique to be directly applied to point clouds containing different and variable noise levels;

- *An iterative grow-merge procedure capable of retrieving connected planar regions with a great level of precision and detail* (Section 3.3). This allows our technique to automatically delimit the detected planar regions;

- *A mechanism to automatically adjust the plane detection parameters in order to fit the local distribution of samples in the dataset* (Sections 3.2.1 to 3.2.3). This lends to great convenience, as users do not need to specify or tune parameter values.

## 2. Related Work

Plane detection in point clouds has been extensively studied, and most techniques can be classified in three broad categories: *Hough transform*, *RANSAC*, and *Region growing*.

### *2.1. Hough Transform*

The *Hough transform* (HT) was proposed to detect lines in binary images [18], and variations have been introduced to improve its efficiency [19], detect planes [20], as well as arbitrary shapes in multidimensional data [21].

The key idea behind the Hough transform is to map input samples to some feature space and cast votes into an accumulator. The *voting process* consists of incrementing the values of all accumulator's cells for the detectable shapes that could contain the given input sample. Detecting shapes then consists of finding peaks in the accumulator, whose resolution depends on the quantization applied to each dimension of the feature space. This process is computationally expensive, becoming prohibitive as the number of dimensions (parameters) increases. Plane detection requires a tridimensional accumulator defined by the parameters $(\theta, \phi, \rho)$, where the angles $(\theta, \phi)$ define the orientation of the plane (*i.e.*, its normal), and $\rho$ is the distance from the plane to the origin. The computational cost of the *standard Hough transform* (SHT) is $O(n\, N_\theta N_\phi)$, where $n = |P|$ is the number of points in the point cloud $P$, and $N_\theta$ and $N_\phi$ are the

number of bins used to discretize angles $\theta$ and $\phi$, respectively. Next, we discuss the main strategies for accelerating plane detection with HT.

The *Probabilistic Hough Transform* (PHT) [22] tries to detect planes using a random subset of the original samples. As such, it is non-deterministic, but its cost is still asymptotically the same SHT's. Moreover, finding the right percentage for the random subset is non-trivial.

The *Randomized Hough Transform* (RHT) casts votes for planes defined by groups of three non-collinear randomly selected samples. Thus, each group casts a single vote, drastically reducing the voting cost. Unfortunately, like PHT, it is also non-deterministic. Thus, there is no guarantee that all planes will be detected, and the results are often not consistent among multiple executions.

Limberger and Oliveira [20] extended the Kernel-based Hough transform [19] for plane detection. It consists of adaptively partitioning the point cloud using an octree, detecting clusters of approximately-coplanar samples in the octree cells using principal component analysis (PCA), and voting for each cluster at once, instead of for individual samples, using a trivariate Gaussian kernel. They achieve state-of-art real-time detection with cost $O(n \log n)$. However, due to PCA, the technique is sensitive to outliers. Moreover, the user may need to adjust the values of the technique's *isotropy* and *isometry* parameters, which depend on the amount of noise in the point cloud.

### *2.2. RANSAC*

*RANdom Sample Consensus* (RANSAC) [23] is another popular technique for shape detection in multidimensional data, due to its simplicity and robustness. RANSAC works iteratively in two steps: first, it draws a minimal random set from the input and estimates a shape (model) from it; then, it evaluates an inlier function for the entire input set. The minimal random set with most inliers is retrieved, along with its estimated shape. This process is repeated to detect multiple shapes. RANSAC's cost for detecting a single plane is $O(I(E + nK)) = O(In)$, where $I$ is the number of iterations used to detect one plane, $E$ is the cost to estimate a plane from three samples, and $K$ is the

cost of checking if a sample lies on the estimated plane.

Schnabel et al. [24] proposed an optimized version of RANSAC using spatial information to only draw nearby samples, and using normal information to improve the inlier function. However, the resulting algorithm is still prone to detecting spurious shapes. Li et al. [25] proposed a pre-processing step for RANSAC intended to prevent the detection of spurious planes. It consists of first subdividing the point cloud into cells, detecting the cells containing coplanar samples and using only those during RANSAC. However, it relies on a good parameter tuning for planar cell detection, which depends on the noise scale of the point cloud. Moreover, this technique is still prone to the detection of spurious planes (see, its result for the *Box* dataset in Figure 8 (RANSAC NDT)). Mittal et al. [26] proposed a new M-Estimator that does not require the specification of an inlier threshold. It estimates the scale of noise by capturing the difference of density between inliers and outliers in multiple random hypotheses. Such technique is prone to merging close planes if the noise scale is underestimated, or to miss planes if it is overestimated. Also, since it requires the analysis of a large number random models to obtain a good estimate of the noise scale, it is slow and unfit for point clouds with millions of samples.

Despite their great flexibility, RANSAC techniques rely on randomness, which may require many iterations to achieve stability, making them slow. Also, for most point clouds captured from real scenes, samples are not uniformly distributed. Thus, planes far from the sensor may never be detected, as they are often undersampled. Another major limitation is the high sensitivity to its parameters, mainly the threshold used in the inlier function, which may be difficult to adjust and depends on the noise level of the point cloud.

### *2.3. Region Growing*

*Region growing* (RG) algorithms are very popular in 2D and 2.5D image applications. Their use in unorganized point clouds, however, is still vastly underrepresented. This is mainly due to the nature of such datasets, which contain occlusions, noise, and lack neighborhood information.

Region growing techniques generally treat the point cloud as a graph, with the samples representing the graph vertices and their spatial proximity (neighborhood) defining the edges. These techniques select seed samples to start a graph search according to an inlier condition (*e.g.*, the angular difference between the normals of neighbor samples should be below a certain threshold). To reduce computational cost, some techniques partition the space containing the point cloud into voxels and perform the graph search on each voxel.

Farid [2] introduced a region growing technique for robotic applications based on a smoothness constraint. Unfortunately, this approach is prone to under-segmentation as the smoothness constraint is often violated.

The work of Pham et al. [27] is similar to ours in the sense that it also uses a partitioning strategy to extract small planar patches. However, it uses a minimization algorithm to extract planes that often merges non-related planes.

Vo et al.'s technique [28] is the closest related to ours. It uses a two-step coarse-to-fine approach. In the first step, an octree is created and on each node it estimates a plane using PCA. Each such plane is used to calculate a residual value obtained as the mean squared distance from the estimated plane to the samples in the octree cell. If the residual is smaller than a threshold, the cell is considered "planar"; otherwise, the cell is recursively subdivided and re-tested until it reaches a minimal size. After this, adjacent "planar cells" are merged if the angular difference between their plane normals is below a certain threshold. Note that this will merge adjacent parallel planes (see, for instance, the technique's result for the *Computer* dataset in Figure 8). In the second step, individual samples belonging to neighbor "non-planar cells" are added to the cell if the distance from the sample to the cell's estimated plane is less than a fixed threshold.

There are, however, key differences between Vo et al.'s technique and ours. First, we use a robust planarity test instead of PCA, which is sensitive to outliers. Second, instead of a top-down partitioning strategy, we use a bottom-up approach, starting from the leaf nodes with minimal size up to the octree root. This strategy proves to be more successful in preserving fine details, while its im-

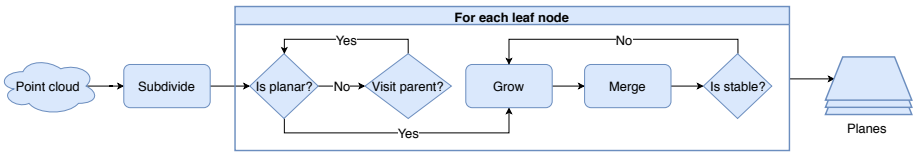


Figure 2: Our robust plane-detection pipeline. An input point cloud is spatially subdivided using an octree until the leaf nodes have less than $\epsilon$ samples. If the samples in the octree cell pass the planarity test, they undergo a growth process and can be merged with adjacent cells into larger planar patches. The output consists of the detected planar regions.

pact on the algorithm's execution time is rather small due to our fast planarity test. Our refinement step is also different. Instead of using fixed thresholds, which require tuning for each different dataset, we estimate them directly from the data distribution.

## 3. Robust Plane Detection

Our region-growing algorithm for plane detection is based on robust statistics and consists of three main steps: *split*, *grow*, and *merge*. In the split phase, the input point cloud is spatially subdivided using an octree[1] until the leaf nodes have less than $\epsilon$ samples. If the samples in an octree leaf node pass our robust-to-noise planarity test, they undergo a growth process and can further be merged with adjacent cells into larger planar patches. In the grow phase, patches that passed the planarity test are augmented in order to fill in possible gaps left by the split phase. However, the growth parameters (maximum distance to the plane and maximum normal deviation) are automatically estimated from the data distribution. Finally, in the merge phase, patches are merged according to some boundary condition. The grow and merge phases are iterated until all patches become *stable*, that is, no further growth is possible. The output of the entire process consists of the set of detected delimited planar regions. A simplified version of this pipeline is shown in Figure 2. Next, we provide the details for these three steps.

[1] A tree data structure where each internal node has exactly eight children.

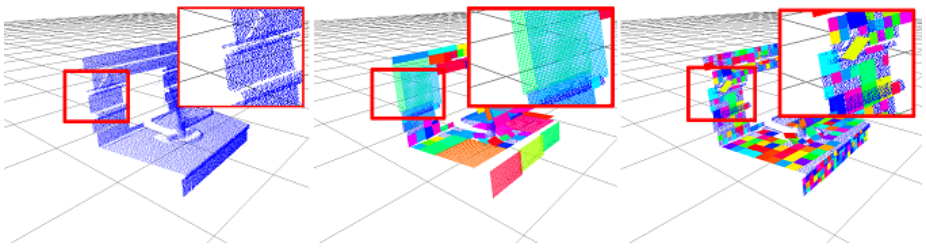

Figure 3: Top-down versus bottom-up approaches to planar patch detection. (left) Original point cloud. (middle) A top-down strategy stops subdividing on the first approximately planar detection, failing to capture small parallel planar regions. (right) Our bottom-up approach keeps subdividing until such structures can be detected.

### *3.1. Split Phase*

Our technique starts by detecting minimal planar regions (patches) in the point cloud. In order to detect such regions, it requires a partitioning technique and a planarity test. An octree containing the entire point cloud is used to recursively subdivide the space until each cell contains less than $\epsilon$ samples, which has been empirically defined as 0.1% of the total number of samples. The planarity test is then applied from the leaves up to the root, stopping whenever a planar patch is detected. Upon detection, the samples move to the grow step (Figure 2). If all eight cells associated to a parent octree node fail the planarity test, the parent node itself becomes the new octree leaf node (replacing its eight children) and undergoes the planarity test. This process is applied recursively. Such a strategy allows the detection of fine details, as shown in Figure 3 (right). The number of detected planar patches is not an issue, since they will be later merged. Algorithm 1 summarizes this process.

### *3.2. Robust Planarity Test*

Most point clouds acquired from real scenes are susceptible to noise, which may be introduced during the acquisition phase or during post-processing. In the acquisition phase, many factors may lend to noise: sensor limitations, the materials present in the scene (*e.g.*, specular surfaces), complex scene geometry (*e.g.*, edges may cause beam splitting), etc. During post-processing, errors in

**Algorithm 1** Detect Planar Patches

**Require:** $O$ {an octree node}, *patches* {set of detected planar patches. An empty set for the octree root call.}

```
1:  procedure DetectPlanarPatches(O, patches)
2:      if #O_points < ε then
3:          return false
4:      O.createChildren()                          ▷ subdivide the octree node
5:      hasPlanarPatch ← false                      ▷ recursively call each child node
6:      for all child ∈ O_children do
7:          if DetectPlanarPatches(child, patches) then
8:              hasPlanarPatch ← true
9:      if not hasPlanarPatch then
10:                 ▷ check the node for planarity only if no child is planar
11:         if RobustPlanarityTest(O_samples) then
12:             removeOutliers(O_samples)
13:                                 ▷ insert node in the list of planar patches
14:             pp.insert(new PlanarPatch(O_samples))
15:             hasPlanarPatch ← true
16:     return hasPlanarPatch
```

registering partial point clouds also lends to noise. It is clear that in order for an algorithm to work well in this scenario, it must be able to handle a considerable amount of noise.

The standard procedure to test co-planarity of a set of samples uses principal component analysis to obtain an eigen-decomposition of the samples' covariance matrix [20, 25, 28]. One way to check for (approximate) co-planarity is by comparing the ratio between the largest and smallest eigenvalue magnitudes. If such ratio is bigger than some threshold $\tau$, the samples can be considered coplanar:

$$|\lambda_1| \leq |\lambda_2| \leq |\lambda_3|, \frac{|\lambda_3|}{|\lambda_1|} > \tau. \tag{1}$$

This procedure, however, has some limitations. The first one is determining the best threshold $\tau$ for a set of samples to be considered approximately coplanar. An ideal threshold should take into consideration the samples' noise level. This, however, is not trivial to determine, since in most point clouds noise is not uniformly distributed, and it is hard to estimate it without knowing the characteristics of the sensor used for acquisition. The second limitation has to

do with the fact that PCA is calculated upon the covariance matrix, which in turn, is calculated using the mean of each variable. Since outliers disturb the mean, they also affect PCA, making it less reliable on noisy datasets. This limitation also applies to covariance-free PCA, because even though it does not use covariance, it is still based on the mean of the observations [29]. Although Robust PCA [30] does not suffer from the outlier problem, it is slow, not being suitable for handling point clouds with millions of samples.

In Robust Statistics [31], *breakdown point* is the percentage of outliers an estimator can handle before giving incorrect results. The mean estimator is said to have a breakdown point of 0%, since a single outlier can disturb it. The median estimator, in turn, has breakdown point of 50%, since it would require more than 50% of outlier observations to disturb it. Thus, the median is said to be a robust alternative to the mean. Likewise, there is a robust alternative to the standard deviation estimator, named *median absolute deviation* ($MAD$). It is obtained as the median of the absolute deviations from the samples' median:

$$MAD = k \times median(|x_i - median(X)|), \tag{2}$$

where $x_i \in X$ represent all individual samples in the set $X$, and $k$ is a constant to make MAD yield consistent results with standard deviation. For a normal distribution, $k = 1.4826$ [32]. Like the median, MAD also has a breakdown point of 50%.

Using these two estimators, we developed a novel planarity test that is robust to noise. It works by first robustly estimating a plane and then checking its planarity using robust tests. A plane $\Pi$ can be defined by a pair $(C, N)$, where $C$ is a point on the plane, and $N$ is the plane's normal. $C$ is estimated as the median of the positions of all samples in the set, while $N$ is a normalized vector whose direction is given by the median of each component of the samples' normals, as shown in Equations 3 and 4:

$$C = [median(S_x), median(S_y), median(S_z)]^T, \tag{3}$$

$$N = \frac{[median(S_{Nx}), median(S_{Ny}), median(S_{Nz})]^T}{||[median(S_{Nx}), median(S_{Ny}), median(S_{Nz})]||}, \quad (4)$$

where $S$ represents the set of samples, $S_k$ and $S_{Nk}$ are, respectively, the $k \in \{x, y, z\}$ component of the position and normal vectors of the samples in $S$. We propose three robust tests to evaluate the quality of the estimated planes: a plane-sample distance test (T1), a plane-sample normal deviation test (T2), and an outlier percentage test (T3).

*3.2.1. Plane-sample distance test*

The first part of the planarity test evaluates the variance in the sample positions relatively to the dimensions of the tested patch. This is required because one can tolerate bigger sample variance in larger patches than in smaller ones. The distance of each sample to the estimated plane $\Pi$ is obtained using Equation 5, defining a new set of observations $D = \{d_1, d_2, ..., d_n\}$:

$$d_i = |(\mathcal{P}_i - C) \cdot N|, \quad (5)$$

where $\mathcal{P}_i \in \Re^3$ is the position of sample $i$, $C$ is the point on the plane computed with Equation 3, and $N$ is the estimated plane normal (Equation 4). Given the set $D$, we compute an interval $I_D$ around the median covering all values which are $\alpha$ MADs from the median:

$$I_D = [median(D) - \alpha MAD(D); median(D) + \alpha MAD(D)]. \quad (6)$$

Such interval has a direct parallel to the *z-score*, which corresponds to the number of standard deviations a value is from the mean. For a normal distribution, the z-score can be used to detect outliers, since the further a value is from the mean, the more unlikely it is to be sampled.

For a normal distribution, 3 standard deviations around the mean correspond to 99.7% of the area under the curve. Since MAD is consistent to standard deviation, 3 MADs also correspond to 99.7% of the area under the curve. Thus, we

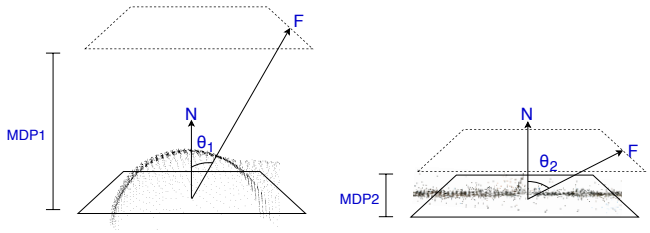


Figure 4: A curved surface produces a large MDP value (left), while a planar surface lends to a small one (right). For same size patches, a larger MDP value result in a smaller angle $\theta$ between F and N (the plane normal).

set $\alpha = 3$ to detect outliers with 99.7% of confidence. Since we are dealing with distances to a plane, the upper limit of this interval represents the maximum distance a sample can be to a plane before it is considered an outlier. We call this upper limit the **maximum distance to plane** (MDP).

When considering a set of approximately coplanar samples, the value MDP = 3×MAD may assume unreasonable large values in the case of datasets containing high levels of noise. Thus, we automatically estimate an *adaptive distance threshold* relative to the dimensions of the patch. Given the plane normal $N$, an orthonormal basis $B = \{U, V, N\}$ for $\Re^3$ can be computed. The actual directions of vectors $U$ and $V$ are not relevant; thus, $U$ and $V$ can be obtained as:

$$U = \frac{[N_y - N_z, -N_x, N_x]^T}{||[N_y - N_z, -N_x, N_x]||}, V = N \times U. \tag{7}$$

Let $F = 0.5\, max(w, l)U + (MDP)N$ be a vector (Figure 3.2.1), where $w$ and $l$ are the estimated planar patch's width and length, respectively. Note that samples from a curved surface results in a large MDP value, while samples from a planar one lends to a small MDP value. For same size patches, large MDP values result in smaller angles between $F$ and $N$ (*i.e.*, $\lim_{MDP \to \infty} \theta = 0°$) (Figure 3.2.1). We take advantage of this observation to evaluate the variance in the sample positions relatively to the patch dimensions as *an angular threshold that works for different point cloud configurations. This eliminates the need for the user to provide or tune parameter values* to improve plane detection for

different point clouds. Larger patches can admit more sample variance (*i.e.*, larger MDP), which is captured by an angular threshold $\theta$. Experimentally, we found that $\theta > 75°$ provides a conservative angular threshold that produces good results for all tested datasets.

*3.2.2. Plane-sample normal deviation test*

This stage of the planarity test evaluates the deviation of the sample normals with respect to the plane's normal $N$. A set of coplanar samples is expected to have relatively homogeneous normal directions, while non-coplanar samples tend to exhibit high variance in their normal directions. Thus, we compute the angle between each sample normal and the plane normal (Equation 8), creating a new set of observations $\Phi = \{\phi_1, \phi_2, ..., \phi_n\}$:

$$\phi_i = acos(|\mathcal{N}_i \cdot N|), \tag{8}$$

where $\mathcal{N}_i$ is the normal of sample $i$, and $N$ is the estimated plane normal. As in test T1, an inlier interval is defined as:

$$I_\Phi = [median(\Phi) - \alpha MAD(\Phi); median(\Phi) + \alpha MAD(\Phi)]. \tag{9}$$

Again, we set $\alpha = 3$ and are only interested in the upper limit of $I_\Phi$. We call this upper limit **maximum normal deviation** (MND), and discard planes with MND above a certain threshold. Unlike MDP, MND does not depend on the patch dimensions. Experimentally we found that MND $< 60°$ provides good results for all tested datasets.

*3.2.3. Outlier percentage test*

For any given patch, if at least 25% of its samples were considered as outliers (*i.e.*, fall outside of either interval $I_D$ or $I_\Phi$), the plane is considered too noisy and thus discarded.

*3.3. Growth Phase*

Due to the partition of the point cloud into octree cells, some co-planar samples may fall just outside of the volume that generated a given planar patch. Also, some patches might have been discarded due to local noise. In either case, it is necessary to grow the detected planar patches to fill in possible gaps and provide a more accurate detection.

A detected planar patch $\mathsf{P}_i$ has three attributes: an estimated plane $\Pi_i$ defined by the pair $(C_i, N_i)$ (Equations 3 and 4); its maximum distance to the plane ($\text{MDP}_i$) (Section 3.2.1); and its maximum normal deviation ($\text{MND}_i$) (Section 3.2.2). In order to grow patches, we define a **neighborhood graph** $G$ for the entire point cloud, where each vertex of the graph represents a sample, and each edge connects two neighbor samples identified using k-nearest neighbors (we use $k = 50$).

Once the neighborhood graph has been created, we perform a breadth-first search, starting from the vertices corresponding to the samples of patch $\mathsf{P}_i$. A visited sample is added to $\mathsf{P}_i$ if: (i) it does not belong to any other patch, and (ii) it meets the inlier condition for $\mathsf{P}_i$.

The **inlier condition** for $\mathsf{P}_i$ must satisfy two criteria: (i) the distance of sample $j$ to $\Pi_i$ should be less than $\text{MDP}_i$ (*i.e.*, $d_j = |(\mathcal{P}_j - C_i) \cdot N_i| < \text{MDP}_i$); and (ii) the deviation between the sample normal and $N_i$ should be less than $\text{MND}_i$ (*i.e.*, $\phi_j = acos(|\mathcal{N}_j \cdot N_i|) < \text{MND}_i$). Unlike previous region growing methods [28, 2], whose inlier conditions rely on fixed thresholds for sample-to-plane distance and normal deviation, our method does not make any assumptions on the scale or noise level of the point cloud, as the computed robust statistics already consider them.

Since a sample may satisfy more than one patch inlier condition, it is necessary to establish some priority among patches. Thus, before starting the growth phase, the patches are sorted by their maximum normal deviations in ascending order. In that way the least noisy patches will grow first, having the opportunity to grow larger. The entire growth phase is summarized in Algorithm 2.

**Algorithm 2** Grow Patches

**Require:** $P$ {point cloud}, $G$ {neighborhood graph}, *patches* {set of detected planar patches}

▷ *Patches only grow with samples from the point cloud that do not belong to any detected plane*
**procedure** GrowPatches($P, G, patches$)
    $SortPatchesByNormalDeviation(patches)$
    $inliers \leftarrow \emptyset$
    **for all** $patch \in patches$ **do**
        ▷ *samples from all detected planes are inliers*
        $inliers.insert(patch_{samples})$
    **for all** $patch \in patches$ **do** ▷ *all detected patches*
        $q \leftarrow patch_{samples}$
        $outliers \leftarrow \emptyset$
        **while** $q \neq \emptyset$ **do**
            $p \leftarrow q.pop()$
            **for all** $n \in G.neighbors(p)$ **do**
                ▷ *all samples that are neighbor to p*
                **if** $n \notin inliers \wedge n \notin outliers$ **then**
                    ▷ *n does not belong to any detected patch*
                    **if** $patch.isInlier(n)$ **then** ▷ *grow patch*
                        $patch_{samples}.insert(n)$
                        $inliers.insert(n)$ ▷ *allow the patch grow from sample*
                        $q.enqueue(n)$
                    **else** ▷ *outlier wrt this specific patch*
                        $outliers.insert(n)$

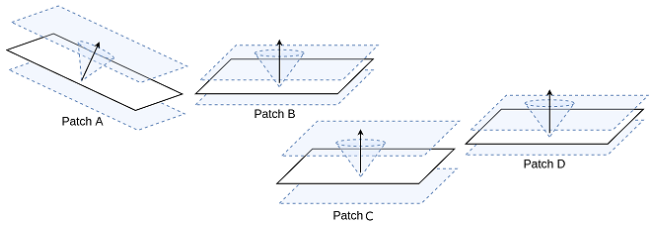


Figure 5: Merging conditions. Patches A and B cannot be merged due to the difference in their normals. A and C cannot be merged because they are not neighbors. Despite having similar normals and being neighbors, B and C cannot be merged because no sample from B or C satisfies the other patch's inlier condition (their distance intervals do not intersect). C and D can be merged.

### *3.4. Merge Phase*

Patches detected at adjacent octree nodes might form a single planar surface and, in this case, should be merged. For two patches $A$ and $B$ to be merged, they must satisfy three **merging conditions**: (M1) they should be adjacent (*i.e.*, the neighborhood graph $G$ should contain at least one edge connecting a sample from $A$ to a sample from $B$); (M2) the estimated normals for $A$ and $B$ should have similar directions (*i.e.*, the angle between these two normals should be less than $\max(\text{MDN}_A, \text{MDN}_B)$); and (M3) at least one sample from $A$ should satisfy the inlier condition for $B$ (or vice-versa). While the first two conditions are self-evident, the third one prevents the merging of parallel patches that are not sufficiently close to each other. These conditions are illustrated in Figure 3.4.

We use a *union-find* data structure to quickly create and find groups of merging patches. The merging process then takes each group of patches that satisfy the three merging conditions and returns a single unified patch.

### *3.5. Iterative Grow-Merge Procedure*

Once a group of patches has been merged, the resulting patch might need to have its associated plane re-estimated, along with its distributions of distances ($D$) and normal deviations ($\Phi$). However, since the breakdown point of our robust estimators (median and MAD) is 50%, an update is only necessary if a patch is extended with more than 50% of its original number of samples. In

such a case, the grow and merge steps (Sections 3.3 and 3.4) need to be re-done, as the inlier condition for the new patch might have changed with respect to its previous state. This iterative process stops when no patch requires update (*i.e.*, when all patches are *stable*). To reduce the number of updates, whenever a group of patches needs to be merged, the patch with most samples is chosen as the group representative.

### *3.6. Complexity Analysis*

Recall that our technique has three main steps: split, grow and merge. The split phase constructs an octree for a set of $n$ samples, whose cost is $O(n\,log_8\,n)$. On each octree leaf, a planarity test is performed. The cost of the planarity test can be broken into: (i) estimating a plane from the median of the samples' positions and normals, whose cost is $O(n)$ [33]; and (ii) obtaining the sets of distance and normal-deviation observations in $O(n)$ time, and then finding the median of these sets, also in $O(n)$ [33]. Thus, the cost of the planarity test is $O(n)$, and the cost of the entire split phase is $O(n\,log_8\,n)$.

For the grow phase, the worst case consists of a single patch with a small number of samples to incorporate all samples in the point cloud. Since no sample is visited more than once, its cost is $O(n)$.

Since no patch can have less than $\epsilon$ samples, a value that we set experimentally to 0.1% of the total amount of samples in the point cloud, the maximum amount of planar patches is constant. Thus, the costs for verifying the three merging conditions are: (M1) $O(n)$; (M2) $O(1)$; and (M3) $O(n)$. The cost of each union-find operations is near constant.

The grow and merge phases are iterated, stopping when all patches become stable. A patch is called *stable* when the number of samples added to it during a grow-merge iteration is less than 50% of its previous number of samples. Thus, the worst case would consist of a single patch starting with $\epsilon$ samples and receiving 50%+1 samples at each iteration. Since the size would be multiplied by 1.5 at each iteration until it reaches $n$ samples (the size of the point cloud), this process might have at most $log_{1.5}(n)$ iterations. Since the cost of each

iteration is $O(n)$, the iterative grow-merge step has cost $O(n \log n)$. Therefore, the total cost of our plane detection algorithm is $O(n \log n)$.

## 4. Results

We implemented our technique in C++ and used OpenGL to render the point clouds and the detected planes. For linear algebra operations, we used the Eigen library [34]. We compared our technique against the most popular as well as the most recent approaches for plane detection, which include methods based on the Hough transform, RANSAC, and region growing. Next, we list the techniques chosen for comparisons and present the reasons for their choices.

The selected Hough-transform-based techniques are the *Randomized Hough Transform* (RHT) [35] and the *3D Kernel-based Hough Transform* (KHT-3D) [20]. RHT is a classic and popular solution for the HT, while KHT-3D is currently the state-of-art real-time plane detection. For RHT, we used the implementation provided by [36], and for KHT-3D we used the implementation provided by the authors [37].

For RANSAC, we chose to compare against the technique of Schnabel et al. [24] and a recent technique by Li et al. [25]. Schnabel et al.'s technique is an efficient RANSAC method for shape detection. The technique of Li et al. [25] is a recent PCA-based solution intended to prevent the selection of spurious planes. It has been chosen to stress the fact that PCA is quite sensitive to noise (*i.e.*, outliers), despite the use of some preprocessing. For both techniques we used the implementations provided by their authors.

For region growing, we selected the techniques of Farid et al. [2], Pham et al. [27], and Vo et al. [28]. Farid et al.'s approach was designed to segment smooth regions in 2.5D depth maps, which we adapted to 3D. Pham et al.'s solution uses a partitioning strategy to extract planar patches. Vo et al.'s recent point-cloud segmentation technique was chosen due to its close resemblance to ours in the sense that it also uses an octree to perform space subdivision and sample clustering. As mentioned in Section 2.3, however, the authors uses

a top-down strategy based on PCA, while we employ a bottom-up solution based on our robust planarity test. Moreover, Vo et al.'s technique uses fixed thresholds that need to be tuned for each dataset to produce the best results. Our technique, on the other hand, does not require any user intervention. Since the source code for Vo et al.'s technique is not available, for the comparisons shown in the paper we used our own implementation of their technique.

To evaluate our technique and compare it to the ones listed above, we used seven datasets, which are shown in Figure 6: (i) *Box*, a cube with 2.5% of uniformly-distributed noise, (ii) *Computer*, a computer desk, (iii) *Room*, (iv) *Utrecht*, the facades of some buildings in the city of Utrecht, (v) *Museum*, (vi) *Plant*, the scanning of a petrochemical plant, and (vii) *Boiler Room*, an industrial boiler room with mostly planes and pipes. Datasets (i) to (v) were used in [20]. Datasets (vi) to (vii) are from Leica's public sample datasets [38]. They were chosen because they span a large number of different characteristics, such as number of samples, sample density, noise level, number of planes, different acquisition sensors, etc. Table 1 presents detailed information about each dataset regarding number of samples, number of planes, and percentage of samples belonging to planar surfaces. With the exception of the Box dataset, which is a synthetic model containing white Gaussian noise, the remaining datasets were captured by real sensors and thus contain natural noise.

Table 1: Dataset details: number of samples, number of planes, and percentage of samples in planar regions.

| | # samples | # planes | % planar regions |
|---|---|---|---|
| Box | 964,806 | 6 | 100 |
| Computer | 68,852 | 9 | 94 |
| Room | 112,586 | 15 | 81 |
| Utrecht | 160,256 | 11 | 70 |
| Museum | 179,744 | 23 | 74 |
| Plant | 358,116 | 9 | 57 |
| Boiler Room | 5,990,481 | 10 | 73 |

### *4.1. Ground Truth*

To evaluate our technique in comparison to others, it was necessary to define some objective metrics computed with respect to the ground truth of each dataset. Since no ground truth was available for these datasets, we created some by interactively selecting the samples that define each plane and using them to estimate the corresponding plane. The selection process was performed using software we developed for such purpose. Each plane was obtained using the median of its sample positions, as well as the median of its sample normals, which were computed using the robust normal estimation technique FAST-MCD [39]. Figure 6 (right) displays the obtained ground truth for all datasets, with each plane in a given dataset shown in a different color.

### *4.2. Evaluation Metrics*

We used three objective metrics adapted from information retrieval to evaluate the detection quality of the compared techniques: *Precision*, *Recall*, and *F1-Score*.

A reliable metric should not just count the number of correctly detected planes, as this may be misleading. Some planes might be more important than others for a given scene. For instance, detecting a large plane should be more important than detecting a small one, since large planes tend to contribute more for the scene reconstruction. However, simply counting the number of samples in the detected planes can also be misleading: surfaces closer to the sensor tend to be more densely sampled than further ones. Such *sensor proximity bias* is illustrated in Figure 7 (left).

To make the number of samples on a planar patch proportional to its area, we *regularize* the point clouds. This process consists of voxelizing the space containing each point cloud and replacing all samples inside a voxel with the voxel centroid. This results in lower-resolution versions of the original point clouds that minimize the proximity-bias effect (Figure 7 (right)). Note, however, that the techniques still perform the actual plane-detection process in the original (*i.e.*, non-regularized) point clouds. The regularized ones are used only for the

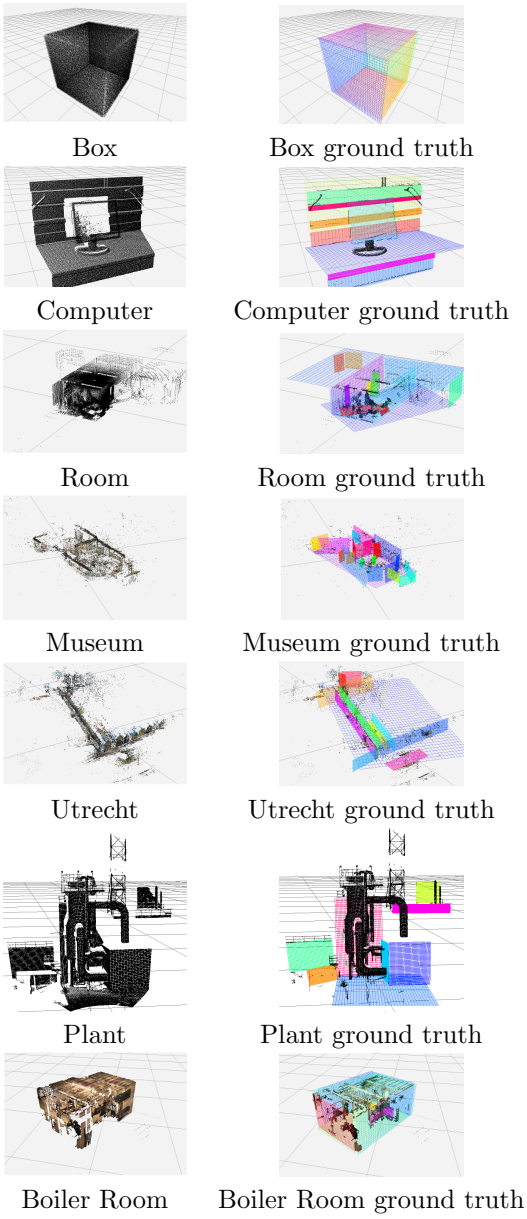


Figure 6: Datasets listed in Table 1. (left) Point clouds. (right) Corresponding ground truths. Each plane is shown in a different color.

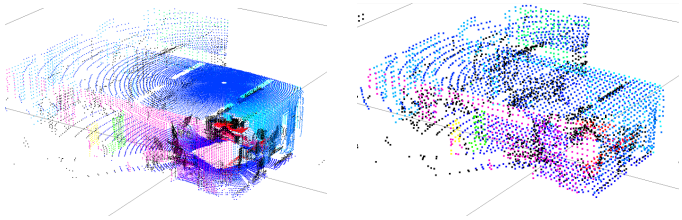

Figure 7: Sensor proximity bias. Planes closer to the sensor are more densely sampled than further ones (left). In order to reduce this bias, we regularize the point cloud, making the number of samples in a planar patch be proportional to its dimensions (right).

purpose of performing quantitative evaluations of the results produced by the compared techniques, as explained next.

The size of each voxel depends on certain properties of the scanning process used to acquire the point cloud (*e.g.*, point clouds acquired through multiple views or scans suffer from less proximity bias than point clouds acquired from a single viewpoint). The relative voxel sizes used for each point cloud are shown in Table 2. They were obtained experimentally to reduce the proximity-bias effect on each dataset and were used to evaluate the results of all techniques.

Given the samples of the original point cloud associated to a given plane $\pi_j$ (either from the ground truth or detected by a technique), if a voxel $v_i$ in the regularized space contains at least one sample belonging to $\pi_j$, that voxel is said to be part of the plane (*i.e.*, $v_i \in \pi_j$). Note that $v_i$ may be part of multiple planes simultaneously. A plane $\pi_{GT_k}$ from the ground truth is considered to be *correctly detected* by a technique $T$ if a plane $\pi_{T_k}$ in T's list of detected planes satisfies the following conditions: (i) at least 50% of the voxels in $\pi_{T_k}$ are also in $\pi_{GT_k}$; and (ii) the angular difference between the normals of $\pi_{T_k}$ and $\pi_{GT_k}$ does not exceed 30°. The first criterion tries to enforce that $\pi_{T_k}$ is centered around the same spatial location as $\pi_{GT_k}$. The second criterion enforces similar orientation, and its threshold was defined empirically.

A voxel $v_i$ is said to be a **true positive** with respect to $\pi_{T_k}$ if it belongs to both $\pi_{T_k}$ and $\pi_{GT_k}$ (*i.e.*, $v_i \in \pi_{T_k}$ and $v_i \in \pi_{GT_k}$), and $\pi_{GT_k}$ was correctly detected by $T$. It is a **false positive** with respect to $\pi_{T_k}$ if $v_i \in \pi_{T_k}$ but

$v_i \notin \pi_{GT_k}$. It is a **true negative** with respect to $\pi_{T_k}$ if $v_i \notin \pi_{T_k}$ and $v_i \notin \pi_{GT_k}$; and it is a **false negative** with respect to $\pi_{T_k}$ if $v_i \notin \pi_{T_k}$ but $v_i \in \pi_{GT_k}$.

Now we can use the classic definitions of precision, recall and F-measure, to evaluate the performance of a technique $T$ for a given dataset $d$ as:

$$\begin{aligned} precision_{T_d} &= \frac{(TP)_{T_d}}{(TP)_{T_d} + (FP)_{T_d}}, \\ recall_{T_d} &= \frac{(TP)_{T_d}}{(TP)_{T_d} + (FN)_{T_d}}, \\ F1_{T_d} &= 2 \times \frac{precision_T \times recall_T}{precision_T + recall_T}, \end{aligned} \tag{10}$$

where $(TP)_{T_d}$, $(FP)_{T_d}$, and $(FN)_{T_d}$ are, respectively, the number of voxels in dataset $d$ corresponding to true positives, false positives, and false negatives, for technique $T$.

Table 2: Voxel sizes used in the regularization of each dataset are relative to the point cloud largest dimension.

| | voxel size (%) |
|---|---|
| Box | 0.02 |
| Computer | 0.02 |
| Room | 0.78 |
| Utrecht | 0.78 |
| Museum | 0.02 |
| Plant | 0.19 |
| Boiler Room | 0.78 |

### *4.3. Time and Quality Analysis*

We ran the experiments comparing the performances of the various techniques on an Intel Core$^{TM}$ i7-7700K 4.20GHz CPU with 32GB RAM. Table 3 summarizes the results for each dataset. To prevent bias, all techniques that require normals (RANSAC (Schnabel) [24], RG (Farid) [2], and RSPD (Ours)) used the same sample normals obtained with FAST-MCD [39], a robust normal estimation technique, using a neighborhood of size 50. The same neighborhoods were used to create the neighborhood graphs used by our technique, RG (Farid) and RG (Vo) [28].

With the exception of our own technique, all other tested ones use fixed threshold parameters. *These have been individually tuned for each specific dataset to produce the best results in each case. Our technique, on the other hand, was applied to all datasets as is.*

Table 3 summarizes the performance of the compared plane detection techniques on the seven datasets listed in Table 1. For each combination of technique and dataset, it shows the number of detected planes (#Detected), as well as the corresponding values for precision, recall, F1-score, and execution time (in seconds). The execution time is an average of 5 runs. The detected planes are shown in Figure 8.

With the exception of RANSAC (NDT) by Li et at. [25], all the techniques performed well on the Box dataset (Figure 8, top row). For RG (Pham), we could not find a set of parameters that yield results in reasonable time. For this dataset, our technique achieved the second highest precision. However, the added noise disturbed the normal estimation for some samples, precluding them from being incorporated into any patches since, in each case, the difference in normal orientation was bigger than MND, and thus decreasing our technique's recall. RG (Vo) achieved the best precision, but a low recall. In this technique, region growing depends on the angular difference among normals of adjacent voxels. Since RG (Vo) estimates voxel normals using PCA, this leads to bad normal estimation near edge regions, which compromises the technique's recall. RANSAC (Schnabel) detected spurious planes on the edges of the Box, mostly due to bended normal estimation in those regions, even using a robust normal estimation technique, due to the high level of noise on the edges. KHT-3D, in turn, achieved the best recall and F-1 score. This simple dataset proved to be an interesting counter-example for the RANSAC (NDT), which performed poorly. RANSAC (NDT) was designed to avoid RANSAC's detection of spurious planes, by discarding voxels containing non-planar regions. Since this dataset only contains planar regions, RANSAC (NDT) reduces to a regular RANSAC.

Most techniques also performed well on the Computer dataset, which is fairly simple. Although our technique detected a spurious plane at the monitor's base,

Table 3: Performance of the evaluated techniques on each dataset. Number of detected planes over total number of planes (#), Precision (P), Recall (R), F1-Score (F1), Elapsed time in seconds (T(s)). Best results highlighted in bold.

| | # | P | R | F1 | T(s) |
|---|---|---|---|---|---|
| **Box** | | | | | |
| RHT | 6/6 | 0.93 | 0.93 | 0.93 | 132.00 |
| KHT-3D | 6/6 | 0.96 | **0.96** | **0.96** | 0.20 |
| RANSAC (Schnabel) | 6/6 | 0.96 | 0.87 | 0.91 | 26.00 |
| RANSAC (NDT) | 5/6 | 0.01 | 0.001 | 0.003 | 67.50 |
| RG (Farid) | 6/6 | 0.95 | 0.91 | 0.93 | 55.60 |
| RG (Pham) | - | - | - | - | - |
| RG (Vo) | 6/6 | **1.0** | 0.58 | 0.73 | **0.13** |
| RSPD (Ours) | 6/6 | 0.97 | 0.84 | 0.90 | 3.35 |
| **Computer** | | | | | |
| RHT | 7/9 | 0.83 | 0.83 | 0.83 | 3.10 |
| KHT-3D | 6/9 | 0.77 | 0.73 | 0.75 | 0.09 |
| RANSAC (Schnabel) | 5/9 | 0.85 | 0.69 | 0.76 | 3.30 |
| RANSAC (NDT) | 6/9 | 0.82 | 0.86 | 0.84 | 3.10 |
| RG (Farid) | 7/9 | **0.98** | 0.76 | 0.85 | 1.18 |
| RG (Pham) | 7/9 | 0.80 | 0.30 | 0.43 | 0.68 |
| RG (Vo) | 5/9 | 0.73 | 0.67 | 0.70 | **0.01** |
| RSPD (Ours) | 9/9 | 0.96 | **0.93** | **0.95** | 0.11 |
| **Room** | | | | | |
| RHT | 6/15 | 0.74 | 0.73 | 0.74 | 25.80 |
| KHT-3D | 5/15 | 0.81 | **0.82** | 0.81 | 0.07 |
| RANSAC (Schnabel) | 5/15 | 0.70 | 0.70 | 0.70 | 3.90 |
| RANSAC (NDT) | 8/15 | **0.95** | 0.39 | 0.56 | 1.08 |
| RG (Farid) | 6/15 | 0.83 | 0.73 | 0.78 | 1.80 |
| RG (Pham) | 5/15 | 0.51 | 0.60 | 0.55 | 0.51 |
| RG (Vo) | 5/15 | 0.58 | 0.60 | 0.59 | **0.03** |
| RSPD (Ours) | 10/15 | 0.85 | 0.81 | **0.83** | 0.24 |
| **Museum** | | | | | |
| RHT | 20/23 | 0.87 | **0.85** | **0.86** | 20.30 |
| KHT-3D | 10/23 | 0.65 | 0.54 | 0.59 | 0.09 |
| RANSAC (Schnabel) | 9/23 | 0.55 | 0.55 | 0.55 | 6.40 |
| RANSAC (NDT) | 16/23 | 0.85 | 0.40 | 0.55 | 21.57 |
| RG (Farid) | 18/23 | **0.94** | 0.75 | 0.84 | 2.50 |
| RG (Pham) | 17/23 | 0.70 | 0.72 | 0.71 | 0.83 |
| RG (Vo) | 14/23 | 0.77 | 0.74 | 0.75 | **0.07** |
| RSPD (Ours) | 17/23 | **0.95** | 0.75 | 0.83 | 0.49 |
| **Utrecht** | | | | | |
| RHT | 7/11 | **0.75** | 0.54 | 0.63 | 31.80 |
| KHT-3D | 4/11 | 0.39 | 0.42 | 0.40 | **0.09** |
| RANSAC (Schnabel) | 3/11 | 0.43 | 0.47 | 0.45 | 5.90 |
| RANSAC (NDT) | 8/11 | 0.73 | 0.31 | 0.43 | 50.40 |
| RG (Farid) | 6/11 | 0.64 | 0.56 | 0.60 | 2.90 |
| RG (Pham) | 10/11 | 0.41 | 0.44 | 0.42 | 4.50 |
| RG (Vo) | 5/11 | 0.62 | 0.46 | 0.53 | 0.21 |
| RSPD (Ours) | 11/11 | **0.75** | **0.74** | **0.74** | 0.44 |
| **Plant** | | | | | |
| RHT | 5/9 | 0.94 | 0.70 | 0.80 | 63.30 |
| KHT-3D | 8/9 | 0.47 | 0.53 | 0.50 | 0.15 |
| RANSAC (Schnabel) | 7/9 | 0.64 | 0.90 | 0.74 | 12.80 |
| RANSAC (NDT) | 8/9 | 0.83 | 0.57 | 0.68 | 2.70 |
| RG (Farid) | 7/9 | **0.97** | 0.76 | **0.85** | 8.60 |
| RG (Pham) | 9/9 | 0.51 | 0.86 | 0.65 | 253.18 |
| RG (Vo) | 5/9 | 0.93 | 0.74 | 0.83 | **0.07** |
| RSPD (Ours) | 9/9 | 0.63 | **0.92** | 0.75 | 0.42 |
| **Boiler Room** | | | | | |
| RHT | 9/10 | 0.74 | **0.84** | **0.78** | 28.60 |
| KHT-3D | 6/10 | 0.63 | 0.83 | 0.72 | **1.15** |
| RANSAC (Schnabel) | 6/10 | 0.77 | 0.68 | 0.72 | 185.40 |
| RANSAC (NDT) | 10/10 | 0.78 | 0.53 | 0.63 | 43.80 |
| RG (Farid) | 6/10 | **0.91** | 0.44 | 0.59 | 368.00 |
| RG (Pham) | 8/10 | 0.54 | 0.77 | 0.64 | 16.04 |
| RG (Vo) | 5/10 | 0.84 | 0.62 | 0.71 | 1.37 |
| RSPD (Ours) | 9/10 | 0.86 | 0.60 | 0.71 | 6.39 |

Table 4: Average performance of the evaluated techniques considering all datasets. Average plane detection ratio (A%), Average precision (AP), Average recall (AR), F1-Score of average precision and average recall (AF1), Average normalized time (normalized time: elapsed time in milliseconds divided by number of samples) (ANT). Techniques sorted by AF-1 score. Best results highlighted in bold.

| | A% | AP | AR | AF1 | ANT |
|---|---|---|---|---|---|
| RANSAC (NDT) | 0.76 | 0.71 | 0.43 | 0.52 | 0.0792 |
| RG (Pham) | 0.75 | 0.57 | 0.61 | 0.56 | 0.1256 |
| KHT-3D | 0.61 | 0.66 | 0.69 | 0.67 | 0.0005 |
| RANSAC (Schnabel) | 0.56 | 0.70 | 0.69 | 0.69 | 0.0355 |
| RG (Vo) | 0.57 | 0.78 | 0.63 | 0.69 | **0.0003** |
| RG (Farid) | 0.69 | **0.88** | 0.70 | 0.77 | 0.0295 |
| RHT | 0.73 | 0.82 | 0.77 | 0.79 | 0.1280 |
| RSPD (Ours) | **0.90** | 0.85 | **0.79** | **0.81** | 0.0021 |

it was able to distinguish nearby parallel planes on the wall, due to our bottom-up partitioning strategy, which granted our method the best recall and F-1 score for that dataset, and the second best precision. As this scene is mostly composed by planar regions, RANSAC (NDT) again detected spurious planes. KHT-3D detected some spurious planes and was not able to extract fine details from the scene. RG (Vo) was not able to extract fine details either. When trying to reduce its voxel size to preserve fine details, spurious planes started to appear.

For the Room dataset, our technique achieved the best F-1 score, and detected most of the planes in the scene (10 out of 15). Most techniques detected spurious planes. Although RANSAC (NDT) obtained the best precision, it also had the worst recall, as it missed many planes.

The Museum dataset contains the largest number of planes (23), most of them small. Therefore, techniques able to analyze connected components and delimit planar regions performed best. Our technique was able to retrieve constrained planes in half of a second. KHT-3D, RANSAC (Schanabel), and RANSAC (NDT) treat all coplanar samples as forming a single plane, regardless of spatial discontinuities. This resulted in elongated planar patches and low precision and F1-score.

The Utrecht dataset is highly noisy, causing most techniques to miss several

planes. Due to our robust planarity test, however, our technique was able to detect all of the existing planes, achieving the best recall, while still detecting each one of them accurately, which also granted it the best precision and F1-score.

The Plant and Boiler Room datasets contain a large number of cylinders and other curved surfaces. Several techniques, including ours, identify approximately planar patches in small cylindrical regions, thus affecting their performance. Nevertheless, our technique achieved good F-1 score on both datasets (0.75 and 0.71, respectively). RG (Farid) performed well in those scenarios, since its region-growing procedure is very sensitive to changes in smoothness. However, this same strategy is prone to false negatives in planar areas containing a high level of noise, as shown in the previous datasets. The performance of RG (Vo) was similar to ours in these two datasets.

Our technique (RSPD) performed well on all datasets, being able to achieve at least the best precision, recall, or F-1 score in 5 of the 7 evaluated datasets, without requiring parameter tuning. For the ones it did not lead the results, it was able to achieve competitive results. On average, it achieved the best overall recall and F-1 score, and the second best overall precision, while being the one of the fastest techniques (Table 4). In terms of time, RG(Vo) and KHT-3D performed extremely well in all datasets, having similar running times.

## 5. Conclusion and Future Work

We presented a robust $O(n \log n)$ technique for detecting planes in unorganized point clouds that achieves better accuracy, measured in terms of average precision, recall, and F1-score, than the previous approaches, while still being one of the fastest. For this, we introduced a novel robust planarity test based on robust statistics that is less sensitive to noise, thus providing a good alternative to the most commonly-used procedure based on PCA. We also introduced an iterative grow-merge strategy capable of detecting delimited planar regions with great level of precision and detail. Our solution is robust to noise and virtually

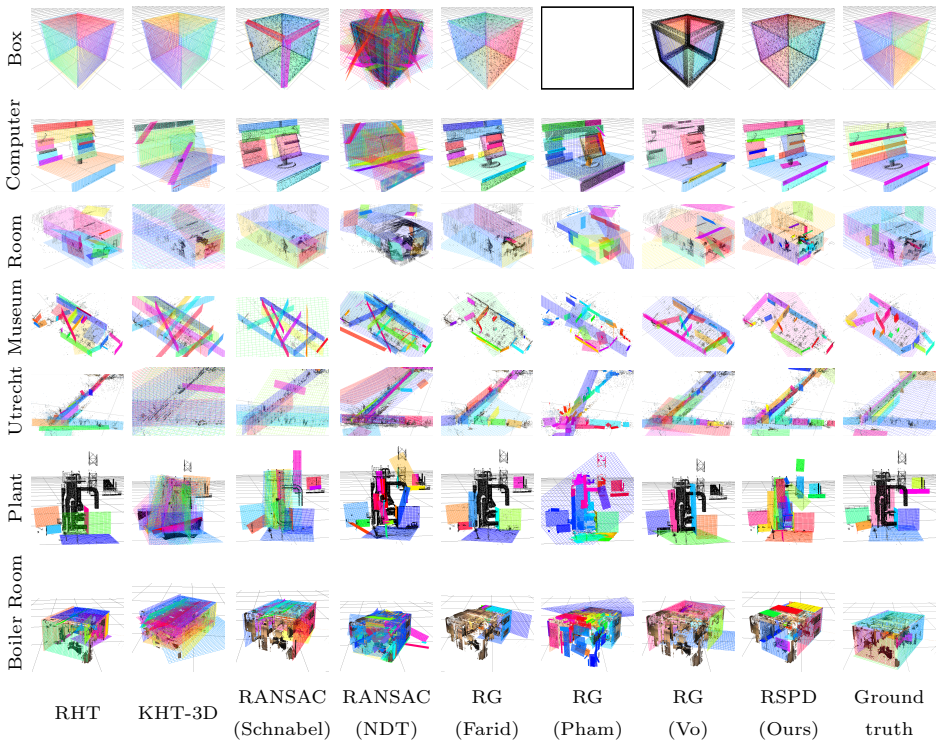


Figure 8: Planes detected by the compared techniques for all datasets. Ground truth is shown in the rightmost column. For each pair of technique and dataset, the detected planes have been highlighted using different colors. Black dots represent samples treated as outliers by each technique. Left-click on the images to zoom in and inspect the details. Larger versions of these images are also available in the supplemental material. We could not find a set of parameters to execute RG (Pham) on the Box dataset in reasonable time.

independent of parameter tunning.

We demonstrated the effectiveness of our technique by performing a detailed comparison with the most popular and with the most recent approaches for plane detection, which include methods based on the Hough transform, RANSAC, and region growing. The techniques were evaluated on seven datasets chosen to cover a large number of different characteristics, such as number of samples, sample density, noise level, number of planes, and different acquisition sensors. Our technique performed well on all datasets, achieving the best average results.

In such an evaluation, the parameters used by all other techniques have been individually tuned for each specific dataset to produce the best results in each case. Our technique, on the other hand, automatically adjusted its parameter values based on the local sample distribution.

Currently, we only perform local planarity tests, disregarding the global structure of the point cloud. This may lend to the detection of false planes on large curved structures (*e.g.*, the detection of rectangular sections along the axis of a cylindrical element with a relatively large radius). This problem could be mitigated by analyzing the surface curvature inside the patch's parent octree node, and preventing the detection of local planar patches in curved surfaces.

## Acknowledgments

This work was sponsored by CNPq-Brazil (fellowships and grants 312975/2018-0, 130895/2017-2, and 423673/2016-5) and by ONR Global Award # N62909-18-1-2131. We thank Li et al. [25], Schnabel et al. [24], Pham et al. [27], Limberger et al. [20] and Borrmann et al. [36] for kindly providing the source code of their techniques.

## References

[1] V. Potó, J. Á. Somogyi, T. Lovas, Á. Barsi, Laser scanned point clouds to support autonomous vehicles, Transportation Research Procedia 27 (2017) 531–537.

[2] R. Farid, Region-growing planar segmentation for robot action planning, in: Australasian Joint Conference on Artificial Intelligence, Springer, 2015, pp. 179–191.

[3] V. Raja, K. J. Fernandes, Reverse engineering: an industrial perspective, Springer Science & Business Media, 2007.

[4] H. Fuchs, Z. M. Kedem, S. P. Uselton, Optimal surface reconstruction from planar contours, Commun. ACM 20 (10) (1977) 693–702.

[5] G. Vosselman, E. Dijkman, 3d building model reconstruction from point clouds and ground plans, Int. Arch. of Photogrammetry and Remote Sensing (2001) 37–43.

[6] R. Kaucic, R. Hartley, N. Dano, Plane-based projective reconstruction, in: Proceedings of Eighth IEEE International Conference on Computer Vision, Vol. 1, 2001, pp. 420–427 vol.1.

[7] F. Tarsha-Kurdi, T. Landes, P. Grussenmeyer, Hough-transform and extended ransac algorithms for automatic detection of 3d building roof planes from lidar data, International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, ISPRS 3 (2007) 407–412.

[8] H. Huang, C. Brenner, M. Sester, 3d building roof reconstruction from point clouds via generative models, in: Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS '11, ACM, 2011, pp. 16–24.

[9] B. Triggs, Autocalibration from planar scenes, in: Proceedings of the 5th European Conference on Computer Vision, Vol. 1 of ECCV '98, Springer-Verlag, 1998, pp. 89–105.

[10] C. A. Rothwell, A. Zisserman, D. A. Forsyth, J. L. Mundy, Planar object recognition using projective shape representation, International Journal of Computer Vision 16 (1) (1995) 57–99.

[11] M. Peternell, T. Steiner, Reconstruction of piecewise planar objects from point clouds, Computer-Aided Design 36 (2003) 333–342.

[12] J. Liu, L. Cao, Z. Li, X. Tang, Plane-based optimization for 3d object reconstruction from single line drawings, IEEE Trans. Pattern Anal. Mach. Intell. 30 (2) (2008) 315–327.

[13] P. Qiu, P. S. Mukherjee, Edge structure preserving 3d image denoising by local surface approximation, IEEE transactions on pattern analysis and machine intelligence 34 (8) (2012) 1457–1468.

[14] G. Simon, A. W. Fitzgibbon, A. Zisserman, Markerless tracking using planar structures in the scene, in: Proceedings of IEEE and ACM International Symposium on Augmented Reality, 2000, pp. 120–128.

[15] D. Chekhlov, A. P. Gee, A. Calway, W. Mayol-Cuevas, Ninja on a plane: Automatic discovery of physical planes for augmented reality using visual slam, in: Proc. of the IEEE and ACM ISMAR, 2007, pp. 1–4.

[16] J. M. Biosca, J. L. Lerma, Unsupervised robust planar segmentation of terrestrial laser scanner point clouds based on fuzzy clustering methods, ISPRS J. of Photogrammetry and Remote Sensing 63 (1) (2008) 84–98.

[17] X. Ning, X. Zhang, Y. Wang, M. Jaeger, Segmentation of architecture shape information from 3d point cloud, in: Proc. 8th Int. Conf. Virtual Reality Continuum and Its Applications in Industry, 2009, pp. 127–132.

[18] P. V. Hough, Method and means for recognizing complex patterns, uS Patent 3,069,654 (Dec. 18 1962).

[19] L. A. F. Fernandes, M. M. Oliveira, Real-time line detection through an improved hough transform voting scheme, Pattern recognition 41 (1) (2008) 299–314.

[20] F. A. Limberger, M. M. Oliveira, Real-time detection of planar regions in unorganized point clouds, Pattern Recognition 48 (6) (2015) 2043–2053.

[21] D. H. Ballard, Generalizing the hough transform to detect arbitrary shapes, Pattern recognition 13 (2) (1981) 111–122.

[22] N. Kiryati, Y. Eldar, A. M. Bruckstein, A probabilistic hough transform, Pattern recognition 24 (4) (1991) 303–316.

[23] M. A. Fischler, R. C. Bolles, A paradigm for model fitting with applications to image analysis and automated cartography (reprinted in readings in computer vision, ed. ma fischler,", Comm. ACM 24 (6) (1981) 381–395.

[24] R. Schnabel, R. Wahl, R. Klein, Efficient ransac for point-cloud shape detection, in: Computer graphics forum, Vol. 26, Wiley Online Library, 2007, pp. 214–226.

[25] L. Li, F. Yang, H. Zhu, D. Li, Y. Li, L. Tang, An improved ransac for 3d point cloud plane segmentation based on normal distribution transformation cells, Remote Sensing 9 (5) (2017) 433.

[26] S. Mittal, S. Anand, P. Meer, Generalized projection-based m-estimator., IEEE transactions on Pattern analysis and machine intelligence 34 (12) (2012) 2351.

[27] T. T. Pham, M. Eich, I. Reid, G. Wyeth, Geometrically consistent plane extraction for dense indoor 3d maps segmentation, in: Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on, IEEE, 2016, pp. 4199–4204.

[28] A.-V. Vo, L. Truong-Hong, D. F. Laefer, M. Bertolotto, Octree-based region growing for point cloud segmentation, ISPRS Journal of Photogrammetry and Remote Sensing 104 (2015) 88–100.

[29] J. Weng, Y. Zhang, W.-S. Hwang, Candid covariance-free incremental principal component analysis, IEEE Transactions on Pattern Analysis and Machine Intelligence 25 (8) (2003) 1034–1040.

[30] J. Wright, A. Ganesh, S. Rao, Y. Peng, Y. Ma, Robust principal component analysis: Exact recovery of corrupted low-rank matrices via convex optimization, in: Advances in neural information processing systems, 2009, pp. 2080–2088.

[31] P. J. Huber, E. M. Ronchetti, Robust Statistics (Wiley Series in Probability and Statistics Book 693), Wiley, 2011.

[32] P. J. Rousseeuw, C. Croux, Alternatives to the median absolute deviation, Journal of the American Statistical association 88 (424) (1993) 1273–1283.

[33] A. Alexandrescu, Fast deterministic selection, in: 16th International Symposium on Experimental Algorithms, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany, 2017, pp. 24:1—-24:18. doi:10.4230/lipics.sea.2017.24.
URL http://drops.dagstuhl.de/opus/volltexte/2017/7612/

[34] Eigen c++ library for linear algebra, http://eigen.tuxfamily.org/index.php?title=Main_Page, accessed: 2018-07-01.

[35] L. Xu, E. Oja, P. Kultanen, A new curve detection method: randomized hough transform (rht), Pattern recognition letters 11 (5) (1990) 331–338.

[36] D. Borrmann, J. Elseberg, K. Lingemann, A. Nüchter, The 3d hough transform for plane detection in point clouds: A review and a new accumulator design, 3D Research 2 (2) (2011) 3.

[37] KHT-3D - real-time detection of planar regions in unorganized point clouds, http://www.inf.ufrgs.br/~oliveira/pubs_files/HT3D/HT3D_page.html, accessed: 2018-12-01.

[38] Leica cyclone/cloudworx example databases, https://hds.leica-geosystems.com/en/29453.htm, accessed: 2018-07-01.

[39] P. J. Rousseeuw, K. V. Driessen, A fast algorithm for the minimum covariance determinant estimator, Technometrics 41 (3) (1999) 212–223.