UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ABNER MATHEUS COSTA DE ARAÚJO

Towards Reverse Engineering of Industrial Site Plants

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Advisor: Prof. Dr. Manuel Menezes de Oliveira
Neto

Porto Alegre
July 2019

CIP — CATALOGING-IN-PUBLICATION

Araújo, Abner Matheus Costa de

Towards Reverse Engineering of Industrial Site Plants / Abner Matheus Costa de Araújo. – Porto Alegre: PPGC da UFRGS, 2019.

81 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2019. Advisor: Manuel Menezes de Oliveira Neto.

1. Point cloud. 2. Reverse engineering. 3. Plane detection. 4. Cylinder detection. I. Menezes de Oliveira Neto, Manuel. II. Título.

“If you wish to make an apple pie from scratch,
you must first invent the universe.”
— Carl Sagan

AGRADECIMENTOS

Agradeço ao meu orientador, sem o qual esse trabalho não teria existido, e ao INF-UFRGS por ter me acolhido durante esses dois anos.

ABSTRACT

CAD models of industrial sites are extremely important, as they provide documentation and simplify inspection, planning, modification, as well as a variety of physical and logistics simulations of the corresponding installations. Despite these clear advantages, many industrial sites do not have CAD models, or have trouble keeping them up-to-date. This is often due to the amount of effort required to create and maintain CAD models updated. Hopefully, the recent popularization of 3D scanning devices is promoting the development of reverse engineering, allowing the creation of 3D representations of real environments from point clouds. Nevertheless, point clouds extracted from industrial sites are extremely complex due to occlusions, noise, non-uniform sampling, size of the dataset, lack of sample organization, among other factors. Thus, a successful reverse engineering solution should have several desirable properties, including speed, robustness to noise, accuracy, and be able to handle point clouds in general without requiring one to fine tune their parameters to each dataset in order to work well on it.

This thesis presents some initial efforts towards obtaining a robust framework for reverse engineering of industrial sites. It introduces two fast and robust algorithms for detecting, respectively, planes and cylinders in noisy unorganized point clouds. Planes and cylinders are typically the most common and largest structures found in those environments, representing walls, floors, ceilings, pipes, and ducts. We demonstrate the effectiveness of the proposed approaches by comparing their performances against the state-of-the-art solutions for plane and cylinder detection in unorganized point clouds. In these experiments, our solutions achieved the best overall accuracy using the same set of (default) parameter values for all evaluated datasets. This is in contrast to the competing techniques, for which their parameter values were individually adjusted for each combination of technique and dataset to achieve their best results in each case, demonstrating the robustness of our algorithms, which do not require fine-tuning to perform well on arbitrary point clouds. Moreover, our technique also displayed competitive speed to other state-of-art techniques, being suitable for handling large-scale point clouds. The thesis also presents a graphical user interface which allows further refinement of the detected structures, providing the user the ability to remove, merge, and semi-automatically detect planes and cylinders in point clouds.

Keywords: Point cloud. Reverse engineering. Plane detection. Cylinder detection.

Em direção à Engenharia Reversa de Plantas Industriais

RESUMO

Modelos CAD de plantas industriais são extremamente importantes, já que eles provêm documentação e simplificam inspeção, planejamento, modificação, assim como uma série de simulações físicas e logísticas das instalações correspondentes. Apesar destas claras vantagens, muitas instalações industriais não contêm modelos CAD, ou têm problemas em mantê-los atualizados. Isto é comumente devido à quantidade de esforço requerida para criar e manter modelos CAD atualizados. Felizmente, a recente popularização de *scanners 3D* está promovendo o desenvolvimento de engenharia reversa, permitindo a criação de representações 3D de ambientes reais a partir de nuvens de pontos. Apesar disto, nuvens de pontos extraídas de instalações industriais são extremamente complexas devido à oclusões, ruído, amostragem não-uniforme, tamanho do conjunto de dados, falta de estrutura das amostras, entre outros fatores. Por isso, uma solução de engenharia reversa de sucesso deveria ter muitas propriedades desejáveis, incluindo velocidade, robustez à ruído, acurácia, e capacidade de lidar com nuvens de pontos em geral sem requerer ao usuário ajustar seus parâmetros para cada conjunto de dados a fim de produzir bons resultados.

Esta tese apresenta alguns esforços iniciais na obtenção de um *framework* de engenharia reversa de plantas industriais. Ela introduz dois algoritmos rápidos e robustos para a detecção de planos e cilindros, respectivamente, em nuvens de pontos não-organizadas com ruído. Planos e cilindros são tipicamente as maiores e mais comuns estruturas encontradas nesses ambientes, representando paredes, chãos, tetos, canos e ductos. Nós demonstramos a eficácia das abordagens propostas comparando suas performances com soluções estado-da-arte para a detecção de planos e cilindros em nuvens de pontos não-organizadas. Nesses experimentos, nossas soluções alcançaram a melhor acurácia em média usando o mesmo conjunto (padrão) de valores de parâmetros para todos os conjuntos de dados avaliados. Isto contrasta com técnicas competidoras, para quais seus parâmetros foram individualmente ajustados para cada combinação de técnica e conjunto de dados a fim de alcançar os melhores resultados em cada caso, demonstrando a robustez de nossos algoritmos, que não requerem um ajuste fino para funcionarem bem em nuvens de pontos arbitrárias. Além disso, nossa técnica também demonstrou velocidade competitiva em relação às outras técnicas do estado-da-arte, sendo adequada para lidar com nuvens

de pontos de larga escala. A tese também apresenta uma interface gráfica que permite o refinamento posterior das estruturas detectadas, provendo ao usuário a habilidade de remover, unir e detectar de forma semi-automática planos e cilindros em nuvens de pontos.

Palavras-chave: Nuvem de pontos, Engenharia reversa, Detecção de planos, Detecção de cilindros.

LIST OF ABBREVIATIONS AND ACRONYMS

CAD	Computer-Aided Design
HT	Hough Transform
RANSAC	RANdom SAmple Consensus
LiDAR	Light Detection and Ranging
PCA	Principal Component Analysis
MAD	Median Absolute Deviation

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

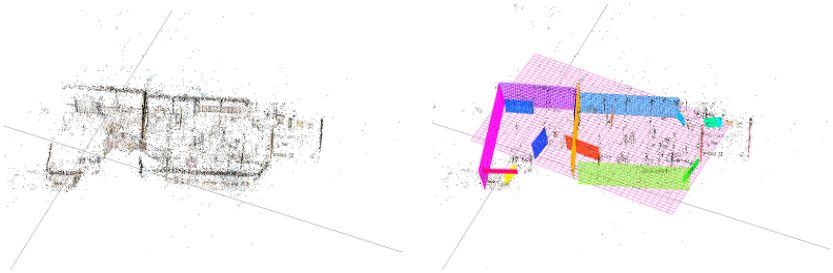
Computer-Aided Design (CAD) is a technology for designing products and documenting the design process. CAD software allows 2D and 3D design with a great level of detail and accuracy. However, many existing installations do not have CAD models or have outdated ones. There are also situations where it is necessary to check if the implementation of a determined project was done according to its design. For these, it would be great if one could use reverse engineering to obtain CAD models directly from point clouds captured from such environments. Hopefully, as *3D scanners* become accurate and accessible, this may become practical.

In industrial sites, planes and cylinders are among the most common structures. They are used to model floors, walls, ceilings, pipes, tanks and ducts. However, detecting even such simple geometric structures in unorganized point clouds is a challenging task. Planes may appear with arbitrary directions and sizes, while cylinders may contain various radii, orientations and lengths. Moreover, unorganized point clouds introduce additional challenges such as noise, non-uniform sampling density, incomplete models due to occlusions, and lack of semantic relationship among samples.

Unfortunately, most available techniques for detecting planes and cylinders in unorganized 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. Regarding cylinder detection, in particular, many techniques make assumptions about expected cylinder radii and orientations (BOLLES; FISCHLER, 1981; LIU et al., 2013; AHMED; HAAS; HAAS, 2014), which significantly restrict their applicability.

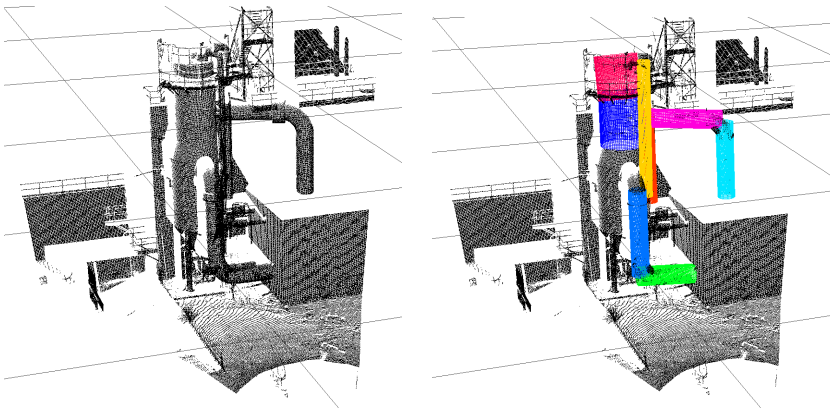
This thesis introduces efficient solutions for detecting planes and cylinders in unorganized point clouds. More specifically, we present an efficient $O(n \log n)$ deterministic technique for plane detection in unorganized point clouds. *Our solution uses robust statistics to derive a novel planarity test that is insensitive to outliers. It uses robust measures of distance to plane and normal deviation to detect and remove outliers, as well as to automatically adjust its 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 plane detection 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

Figure 1.1 – 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.



Source: the author.

Figure 1.2 – Example of automatic cylinder detection using our technique. (left) Point cloud of an actual petrochemical plant. (right) Detected cylinders shown as highlighted polygonal meshes.



Source: the author.

process (Section 4.2). 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 4.4). 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, ours does not require user-specified parameters.

The thesis also presents a fast cylinder-detection technique that is robust to noise, uses parameters which require little to no fine-tuning, and can handle cylinders with arbitrary orientations. *It works by projecting the point cloud onto a set of uniformly-distributed directions defined over the unit hemisphere.* Only samples whose normals are approximately perpendicular to a given direction are projected along such direction. *It then refines these directions and detects circular projections formed by samples defining*

connected components in 3D. The extracted cylindrical surfaces are obtained by fitting a cylinder to each connected component that passes a validity test.

We demonstrated the effectiveness of our plane and cylinder detection solutions by performing detailed comparisons with the most popular, as well as with the most recent approaches for each category. For the 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 1.1 illustrates the use of our plane detection 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.

For our cylinder detection, the comparions were carried out on five datasets, three of which were acquired from real industrial sites. In these experiments, our method achieved the best overall accuracy using the *same set of (default) parameter values for all evaluated datasets*. This is in contrast to the other techniques, for which their parameter values were individually adjusted for each combination of technique and dataset to achieve their best results in each case. This demonstrates the robustness of our approach, which does not require fine tuning to perform well on arbitrary point clouds. Figure 1.2 illustrates the use of our technique applied to a point cloud of a petrochemical plant containing cylinders with various orientations, lengths, diameters, and positions.

1.1 Contributions

This thesis presents the following contributions:

- *A technique for detecting planes in unorganized point clouds that is robust to noise and virtually independent of parameter tunning* (Chapter 4). 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 4.3). 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 4.4). 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 4.3.1 to 4.3.3). This lends to great convenience, as users do not need to specify or tune parameter values;
- A fast technique for cylinder detection in unorganized point clouds that is robust to noise and can handle cylinders with arbitrary orientations (Chapter 5). Its parameter values require little to no fine-tuning to work well with general point clouds;
- A deterministic circle-recognition technique capable of filtering noisy samples (Section 5.5 and 5.6). It is faster than traditional alternatives such as Hough transform and RANSAC.

1.2 Thesis statement

It is possible to use robust statistics to create a planarity test which is robust to noise and presents a linear cost in the number of samples. Futhermore, it is possible to obtain more accurate plane detection results using an octree to visit the samples in a bottom-up approach. Moreover, it is possible to efficiently detect cylinders with arbitrary orientations using a fast procedure to detect circles based on checking whether the sample normals intersect at some point.

1.3 Structure of this thesis

The remaining of this thesis is organized as follows: Chapter 2 discusses some background knowledge necessary for understanding the techniques introduced in this thesis; Chapter 3 discusses related works; Chapter 4 presents our automatic plane detection technique. Specifically, in Section 4.2 we present our bottom-up split strategy in order to increase accuracy; Section 4.3 presents our novel robust planarity test; From Section 4.4 to 4.6 presents our iterative grow-merge procedure; Section 4.7 discusses the complexity analysis of our technique. The results for our plane detection technique are described in Section 4.9, while we discuss the results in Section 4.10; Chapter 5 presents our automatic cylinder detection technique. Sections 5.2 to 5.9 present its details; Section 5.10

discusses its computational cost; Section 5.11 presents the results of our cylinder detection technique which are further discussed in Section 5.12. Chapter 6 presents a graphical user interface for semi-automatic segmentation of planes and cylinders. Finally, Chapter 7 presents some conclusions, limitations, and directions for future work.

2 BACKGROUND ON POINT CLOUDS

This chapter presents some background information about point clouds required for understanding the techniques introduced in this thesis.

3D scanning devices yield 3D representations of the sampled scenes. However, depending on the device, these representations may vary. The most common type of output format is a *point cloud*. A point cloud is a set of 3D samples, which may contain additional per-sample information, such as color, reflected laser intensity, and normal vectors. Figure 2.1 shows an example of a point cloud.

Point clouds can be classified as *organized* and *unorganized*. In an organized point cloud, some neighborhood information is available for each sample. In turn, in unorganized point clouds each sample is completely unrelated to the others.

The coordinates of the samples in a point cloud are defined with respect to the scanner coordinate system, whose origin is centered at the scanner position. However, a complex scene requires multiple scans. Thus, the resulting point clouds need to be brought to the same coordinate system. This process is called *registration*. The most used registration algorithm is the *Iterative Closest Point* (ICP) (BESL; MCKAY, 1992), which uses a reference point to align the individual point clouds.

2.1 Pre-processing algorithms and data-structures

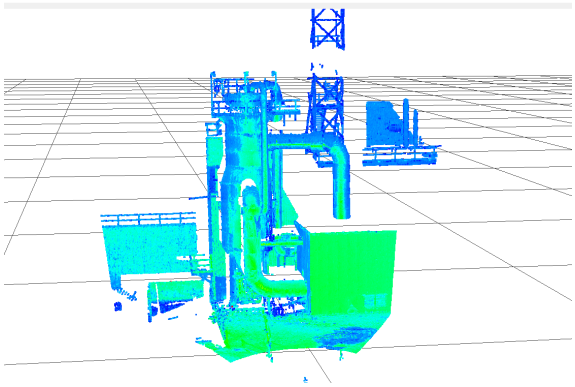
Most point clouds are found in raw format, *e.g.*, containing only the information about the position of the samples. Thus, it is necessary to augment them with meaningful information in order to facilitate the reverse engineering process. This section describes the techniques and data-structures used in this thesis to extract information from unorganized point clouds.

2.1.1 Point-cloud partitioning

An *octree* is a tree data-structure used for spatial partitioning. Each octree node represents a cubic volume, and each intermediary node of the octree contains exactly eight children, where each child represents an octant of the parent cube.

When used to partition a point cloud, any criterion can be employed to stop the

Figure 2.1 – Example of a point cloud, with reflected laser intensities displayed using a heatmap color scale.



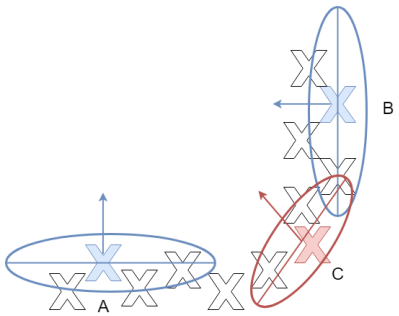
Source: the author.

octree subdivision process. One of the most common is the number of samples which are within the node. In this case, if the number of samples is below a certain threshold, the subdivision process stops. The analogous of the octree for the 2D space is called *quadtree*. It works in similar way, but using square regions instead of cubic volumes. An octree will be used in Section 4.2, while a quadtree will be used in Section 5.5.

2.1.2 Neighborhood graph for point clouds

A neighborhood graph is built using the point cloud samples as vertices and the neighborhood relation among them as the edges. In order to find the neighbors for a given sample, we perform a k-nearest neighbors (KNN) search. The neighborhood graph is used during many steps of our plane and cylinder detection techniques (Chapters 4 and 5, respectively). A naive approach to construct k-nearest neighborhoods consists of calculating the distance from a sample to all the others in the point cloud, and then sorting them by distance. The computational cost of calculating KNN using such brute-force solution is quadratic on the number of samples (*i.e.*, $O(n^2)$, for n samples). A better approach is to use an octree as an auxiliary data-structure and to visit first the leaf node which contains the samples to which the neighbors will be calculated, and then move to the parent nodes to find more neighbors. This algorithm is described in (MOORE, 1991). The computational cost of finding the nearest neighbors using an octree for all samples in the point cloud $O(nlog(n))$, which provides a substantial improvement compared to the brute-force alternative.

Figure 2.2 – The estimation of normals via PCA assumes that the neighborhoods are locally planar. This assumption holds for neighborhoods A and B, but not for C, generating a "bent" normal in this case.



Source: the author.

2.1.3 Normal estimation for samples in point clouds

The normal vector for each sample is important for the reverse engineering process because it provides a local estimate for the surface orientation. However, very rarely normals are present in unorganized point clouds. Thus, they need to be estimated. In its most basic form, normal estimation is done via *principal component analysis* (PCA) in a neighborhood around the sample for which one wants to estimate the normal vector (HOPPE et al., 1992).

If a set of samples is (approximately) planar, the axis with the least variation will be treated as the plane normal. Thus, the use of PCA to estimate normals assumes that the neighborhood is locally planar. Of course, this assumption does not always hold, especially on hard edges, generating bad estimates and "bent" normals, as illustrated in Figure 2.2. To minimize this effect, we estimate sample normals using a more robust technique, called *minimum covariance determinant* (MCD), proposed by Rousseeuw et. al ((ROUSSEEUW; DRIESSEN, 1999)). The goal of MCD for normal estimation is to find samples whose covariance matrix has the determinant with the smallest magnitude among all possible subsets with the same size. Such a subset is the one most suitable to display a local planar property.

MDC works by performing N trials, each using a number of samples equal to half of the neighborhood size. For each trial, MCD calculates the determinant of the covariance matrix of the sample positions. The samples from the trial whose determinant of the covariance matrix has the smallest magnitude is then retrieved and used as the final neighborhood for standard normal estimation via PCA.

3 RELATED WORK

This chapter discusses the main techniques which have been proposed for detecting planes and cylinders in unorganized point clouds.

3.1 Plane detection

Plane detection in point clouds has been extensively studied, and most techniques can be fit in three broad categories: *Hough Transform*, *RANSAC* and *Region growing*. This section discusses the characteristics of each one of those categories.

3.1.1 Hough Transform

The *Hough transform* (HT) was proposed to detect lines in binary images (HOUGH, 1962), and variations have been introduced to improve its efficiency (FERNANDES; OLIVEIRA, 2008), detect planes (LIMBERGER; OLIVEIRA, 2015), as well as arbitrary shapes in multidimensional data (BALLARD, 1981).

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 (θ, ϕ, ρ), where the angles (θ, ϕ) define the orientation of the plane (*i.e.*, its normal), and ρ 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_θ and N_ϕ are the number of bins used to discretize angles θ and ϕ, respectively. Next, we discuss the main strategies for accelerating plane detection with HT.

The *Probabilistic Hough Transform* (PHT) (KIRYATI; ELDAR; BRUCKSTEIN, 1991) 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 (LIMBERGER; OLIVEIRA, 2015) extended the Kernel-based Hough transform (FERNANDES; OLIVEIRA, 2008) 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.

3.1.2 RANSAC

RANdom Sample Consensus (RANSAC) (FISCHLER; BOLLES, 1981) 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. (SCHNABEL; WAHL; KLEIN, 2007) 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. (LI et al., 2017) 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 4.8 (RANSAC NDT)). Mittal et al. (MITTAL; ANAND; MEER, 2012) 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.

3.1.3 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 (FARID, 2015) introduced a region growing technique for robotic applications based on a smoothness constraint. Unfortunately, this approach is prone to undersegmentation as the smoothness constraint is often violated.

The work of Pham et al. (PHAM et al., 2016) 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 (VO et al., 2015) 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 4.8). 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 impact 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.

3.2 Cylinder detection

Like plane detection, cylinder detection can be fit in three broad categories: (i) Hough Transform, (ii) RANSAC and (iii) Region Growing.

3.2.1 Hough Transform

Cylinders can be minimally characterized by five parameters: axis (θ,ϕ), radius (r), and center (C_x, C_y), this last one corresponding to the projection of the axis on a plane

perpendicular to it. Such parameters create a 5D feature space which is both memory- and computationally-prohibitive. As a result, direct application of the Hough transform for cylinder detection is not practical.

Rabbani et al. proposed a two-step Hough transform to mitigate this restriction (RABBANI; HEUVEL, 2005). First, they map the sample normals to a Gauss map. Since the normals of a cylinder form a great circle on the Gauss map, the authors proceed by detecting the planes containing these circles, followed by the projection of the associated samples onto the corresponding detected planes. For each plane, its normal is used as an estimate of a cylinder axis, while the projected circle is used to estimate the cylinder's radius and center. The detections of both planes and circles use Hough transforms. A downside of this technique is that errors during the plane detection step are propagated to the circle detection. Moreover, one needs to adjust parameters for two distinct Hough transforms to achieve good results, which is cumbersome.

Ahmed et al. re-sample the point cloud by slicing it using pre-determined intervals along the X, Y and Z directions, and on each slice they use a Hough transform to detect circles using the projections of the samples falling inside each interval (AHMED; HAAS; HAAS, 2014). Circles with similar centers along the same slicing direction are merged to obtain a cylinder. The advantage of this technique is that it removes the need of a plane detection technique on the Gauss map. However, it is restricted to cylinders aligned with the X, Y, or Z directions.

3.2.2 RANSAC

Bolles and Fischler used RANSAC to detect cylinders with known radius and orientation in range data (BOLLES; FISCHLER, 1981). For each scanline, the technique tries to detect an ellipse and its center. A second RANSAC is used to detect lines in 3D passing through the centers of the detected ellipses, thus obtaining a cylindrical volume.

Chaperon and Goulette (CHAPERON; GOULETTE, 2001) used a two-step RANSAC to detect cylinders in unorganized point clouds. First, they detect planes on a Gauss map and then detect circles on the projection of the point cloud onto the planes found in the first step. This is conceptually similar to and suffers from the same limitations as Rabbani et al.'s (RABBANI; HEUVEL, 2005), but was proposed earlier.

Schnabel et al. proposed a RANSAC technique to detect cylinders, among other geometric primitives (SCHNABEL; WAHL; KLEIN, 2007). To estimate a cylinder, given

two samples, the vector resulting from the cross product of these samples' normals is used as the cylinder axis estimate. They then project these samples and their normals onto the plane perpendicular to the cylinder axis. The projected samples are used to estimate the cylinder's radius, whose center is obtained by extending the projected normals positioned at the projected samples. Albeit presenting good results, the technique suffers from limitations inherent to RANSAC. Thus, it may require many iterations to converge, is non-deterministic, and it may be difficult to detect underrepresented shapes. This later problem is accentuated in point clouds with non-uniform distributions, which is the case of most point clouds obtained from real environments.

Liu et al. (LIU et al., 2013) proposed a RANSAC technique to detect cylinders in pipeline plants which is similar to the work of Chaperon and Goulette (CHAPERON; GOULETTE, 2001). It, however, assumes that every cylinder is either orthogonal or parallel to the ground.

3.2.3 Region Growing

Tran et al. (TRAN; CAO; LAURENDEAU, 2015) used curvature information to select potential seed sample candidates. For each candidate, the technique selects a neighborhood around it and performs an iterative fitting step. This consists of using principal component analysis (PCA) to detect the cylinder axis, and a circle fitting procedure to detect the cylinder's center and radius on the projection of the samples. Samples with good fitting to the estimated parameters are used as seed samples in the next iteration. This process stops after a pre-determined number of iterations. This technique relies on good curvature estimation, which is not trivial to obtain. Moreover, since the iterative fitting procedure is performed for each sample found in high-curvature regions, it is computationally expensive.

Unlike previous methods, our cylinder detection technique does not rely on detecting planes before the actual cylinder detection, and does not make any assumptions about cylinder radius or orientation.

4 PLANE DETECTION

This chapter presents our automatic plane detection technique. In industrial sites, planes represent the ground, walls, ceilings and many other planar structures, which are among the largest and most common structures in those environments.

4.1 Overview

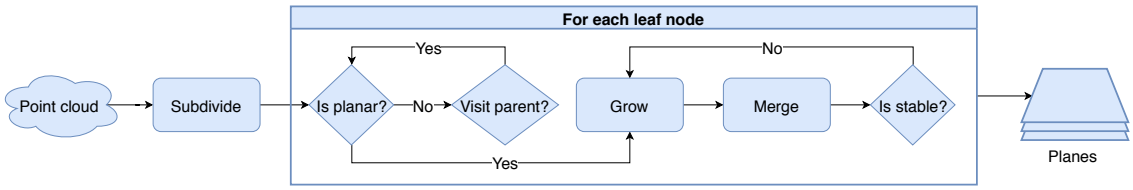
This section presents the overview of our automatic detection technique. We propose a region growing algorithm for plane detection based on robust statistics. It works in three main steps: split, grow and merge.

In the split phase, the input point cloud is spatially subdivided using an octree until the leaf nodes have less than ϵ 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.

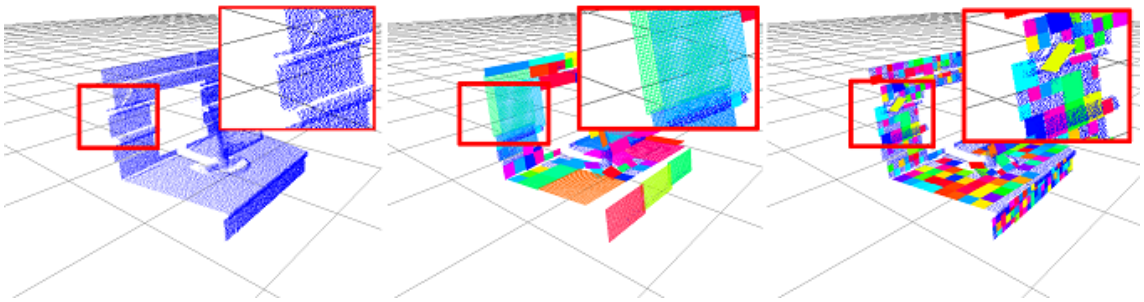
Finally, in the merge phase, patches are merged according to some boundary condition. The grow and merge phases are iterated until all patches become *stable*, that is, no further growth is possible. The output of the entire process consists of the set of detected delimited planar regions. A simplified version of this pipeline is shown in Figure 4.1. Next, we provide the details for these three steps.

Figure 4.1 – Our automatic plane-detection pipeline. An input point cloud is spatially subdivided using an octree until the leaf nodes have less than ϵ 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.



Source: the author.

Figure 4.2 – 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.



Source: the author.

4.2 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 ϵ 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 4.1). 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 4.2 (right). The number of detected planar patches is not an issue, since they will be later merged. Algorithm 1 summarizes this process.

4.3 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 registering partial point clouds also lends

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:             patches.insert(new PlanarPatch(O_samples))
15:             hasPlanarPatch ← true
16:     return hasPlanarPatch
```

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 (LIMBERGER; OLIVEIRA, 2015; LI et al., 2017; VO et al., 2015). One way to check for (approximate) co-planarity is by comparing the ratio between the smallest and largest eigenvalue magnitudes. If such ratio is smaller than some threshold ϵ, the samples can be considered coplanar:

$$|\lambda_1| \leq |\lambda_2| \leq |\lambda_3|, \frac{|\lambda_1|}{|\lambda_3|} < \epsilon. \tag{4.1}$$

This procedure, however, has some limitations. The first one is determining the best threshold ϵ 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 (WENG; ZHANG; HWANG, 2003). Although Robust PCA (WRIGHT et al., 2009) does not suffer from the outlier problem, it is slow, not being suitable for handling point clouds with millions of samples.

In Robust Statistics (HUBER; RONCHETTI, 2011), *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:

$$MAD = k \times median(|x_i - median(X)|), \tag{4.2}$$

where $k = 1.4826$, in order to be consistent with the standard deviation for a normal distribution (ROUSSEEUW; CROUX, 1993).

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 Π 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 4.3 and 4.4:

$$C = [median(S_x), median(S_y), median(S_z)]^T, \tag{4.3}$$

$$N = \frac{[median(S_{Nx}), median(S_{Ny}), median(S_{Nz})]^T}{||[median(S_{Nx}), median(S_{Ny}), median(S_{Nz})]||}, \tag{4.4}$$

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).

4.3.1 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 Π is obtained using Equation 4.5, defining a new set of observations $D = \{d_1, d_2, ..., d_n\}$:

$$d_i = |(\mathcal{P}_i - C) \cdot N|, \tag{4.5}$$

where $\mathcal{P}_i \in \Re^3$ is the position of sample i, C is the point on the plane computed with Equation 4.3, and N is the estimated plane normal (Equation 4.4).

Given the set D, we compute an interval I_D around the median covering all values which are α MADs from the median:

$$I_D = [median(D) - \alpha MAD(D); median(D) + \alpha MAD(D)]. \tag{4.6}$$

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.

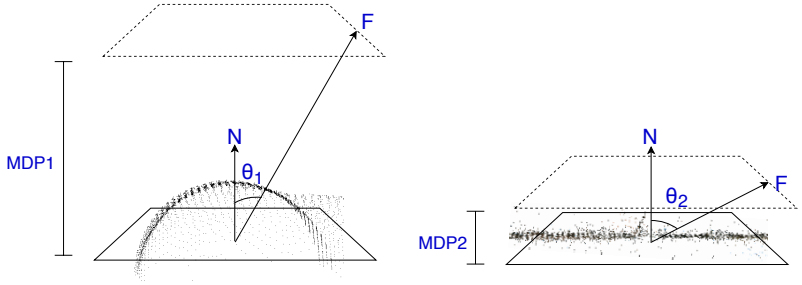
For a normal distribution, 3 standard deviations around the mean correspond to 99.7% of the area under the curve. Since MAD is consistent to standard deviation, 3 MADs also correspond to 99.7% of the area under the curve. Thus, we set $\alpha = 3$ to detect outliers with 99.7% of confidence.

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

Figure 4.3 – 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 θ between F and N (the plane normal).



Source: the author.

be obtained as:

$$U = \frac{[N_y - N_z, -N_x, N_x]^T}{||[N_y - N_z, -N_x, N_x]||}, V = N \times U. \tag{4.7}$$

Let $F = 0.5\, max(w, l)U + (MDP)N$ be a vector (Figure 4.3), where w and l are the estimated planar patch's width and length, respectively (computed as the dimensions of the bounding box of the projection).

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 4.3).

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 θ. Experimentally, we found that $\theta > 75°$ provides a conservative angular threshold that produces good results for all tested datasets.

4.3.2 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 4.8), creating a new set of observations $\Phi = \{\phi_1, \phi_2, ..., \phi_n\}$:

$$\phi_i = acos(|\mathcal{N}_i \cdot N|), \tag{4.8}$$

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:

$$I_\Phi = [median(\Phi) - \alpha MAD(\Phi); median(\Phi) + \alpha MAD(\Phi)]. \tag{4.9}$$

Again, we set $\alpha = 3$ and are only interested in the upper limit of I_Φ. 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 $< 60^\circ$ provides good results for all tested datasets. Note that ϕ_i is between 0^{circ} and 90^{circ} (since we take the module of the dot product). It is theoretically possible that the MND exceeds the 360^{circ} (which would result in cycling back to 0^{circ}). However, since no angle can exceed the 90^{circ} threshold, in practical terms, we can cap the MND to 90^{circ} max.

4.3.3 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_Φ), the plane is considered too noisy and thus discarded.

4.4 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 P_i has three attributes: an estimated plane Π_i defined by the pair (C_i, N_i) (Equations 4.3 and 4.4); its maximum distance to the plane (MDP$_i$) (Section 4.3.1); and its maximum normal deviation (MND$_i$) (Section 4.3.2). In order to

grow patches, we use the **neighborhood graph** G, which was defined in Section 2.1.2, with a neighborhood of size $k = 50$.

We then perform a breadth-first search, starting from the vertices corresponding to the samples of patch P_i. A visited sample is added to P_i if: (i) it does not belong to any other patch, and (ii) it meets the inlier condition for P_i.

The **inlier condition** for P_i must satisfy two criteria: (i) the distance of sample j to Π_i should be less than 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 MND_i (*i.e.*, $\phi_j = acos(|\mathcal{N}_j \cdot N_i|) < \text{MND}_i$).

Unlike previous region growing methods (VO et al., 2015; FARID, 2015), 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.

4.5 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 4.4.

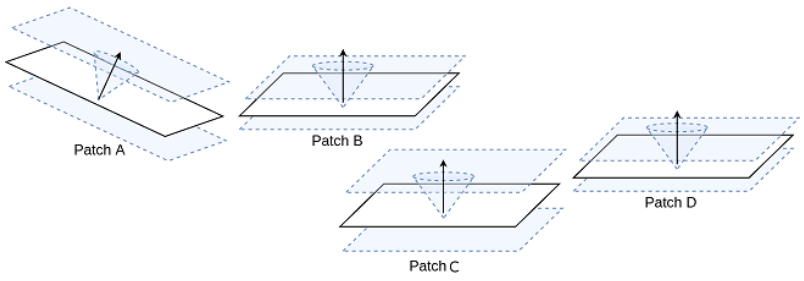
We use a *union-find* data structure to quickly create and find groups of merging patches. The merging process then takes each group of patches that satisfy the three merging conditions and returns a single unified patch.

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 4.4 – 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.



Source: the author.

4.6 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 (Φ). 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 4.4 and 4.5) 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.

4.7 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 $O(n)$ (ALEXANDRESCU, 2017); 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 $O(n)$ (ALEXANDRESCU, 2017). 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 ϵ 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 ϵ samples and receiving over 50% of 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 automatic plane detection algorithm is $O(n\,log\,n)$.

4.8 Plane Delimitation

After the planar regions have been properly detected, it is interesting to delimit a plane geometry. This is particularly interesting in fields like Reverse Engineering where the goal is to extract a set of primitives which can be analyzed by CAD softwares.

A delimited plane is composed by a orthogonal basis $[U, V, N]$, where N is the plane normal and U and V are the main directions which define the plane orientation. U and V are scaled by constants w and l, respectively, to represent the plane width and length.

In order to find the proper U and V directions, along with w and l, first we project all points within a plane region onto any orthogonal base where one the axes is the plane normal. Equation 4.7 displays an example. The projection results in a new 2D space, with new coordinate system (u, v), as shown in Equation 4.10.

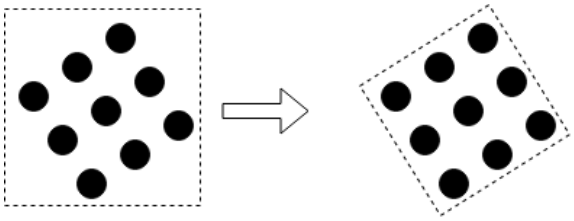
$$P(u, v) = [P(x, y, z) \cdot U, P(x, y, z) \cdot V]^T \tag{4.10}$$

After the projection, we compute the *Convex Hull* of the projected points, returning a subset of points S. On S, we calculate a bounding box R with area a, as described in Equation 4.11.

$$a = (max(S_u) - min(S_u)) \times (max(S_v) - min(S_v)) \tag{4.11}$$

The goal is to find an orthogonal basis $[U, V, N]$ which results in the minimum bounding box area, thus resulting in the best-fit rect. We can find it by rotating the projections by a certain amount of degrees between $[0°, 90°]$ and finding the rotation degree which results in the minimum bounding box area. However, instead of performing a linear search of the rotation interval, we can perform a binary search, reducing the number of rotations drastically, as shown in Algorithm 3.

Figure 4.5 – Delimiting the plane is equivalent to finding an orthogonal basis whose bounding box is mimimal. The left bounding box has more area than the right bounding box, thus it is not a good candidate for the plane bounding box.



Source: the author.

Algorithm 3 Delimit Planes

Require: P {point cloud} G {neighborhood graph} $patches$ {set of patches}

procedure DELIMITPLANES($P, G, patches$)
  $planes \leftarrow \emptyset$
  for all $patch \in patches$ **do**
    $N \leftarrow patch_{normal}$
    $U \leftarrow [N_y - N_z, -N_x, N_x]^T$
    $V \leftarrow \frac{N \times U}{||N \times U||}$
    $S \leftarrow \emptyset$
    for all $p \in patch_{points}$ **do**
      $S.insert([P[p]_{point} \cdot U, P[p]_{point} \cdot V]^T)$
    $S \leftarrow ConvexHull(S)$
    $\theta_{min} \leftarrow 0$
    $\theta_{max} \leftarrow 90$
    while $\theta_{max} - \theta_{min} > \alpha$ **do**
      $\theta_{mid} \leftarrow (\theta_{min} + \theta_{max})/2$
      $\theta_{left} \leftarrow (\theta_{min} + \theta_{mid})/2$
      $\theta_{right} \leftarrow (\theta_{mid} + \theta_{max})/2$
      if $CalculateArea(S, \theta_{left}) < CalculateArea(S, \theta_{right})$ **then**
        $\theta_{max} \leftarrow \theta_{mid}$
      else
        $\theta_{min} \leftarrow \theta_{mid}$
    $U \leftarrow RotateAroundAxis(U, N, \theta_{mid})$
    $V \leftarrow RotateAroundAxis(V, N, \theta_{mid})$
    $w \leftarrow GetWidth(S, U)$
    $l \leftarrow GetLength(S, V)$
    $planes.insert(newPlane(U, V, N, w, l))$
  return $planes$

4.9 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 (EIGEN...,). 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) (XU; OJA; KULTANEN, 1990) and the *3D Kernel-based Hough Transform* (KHT-3D) (LIMBERGER; OLIVEIRA, 2015). 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 (BORRMANN et al., 2011), and for KHT-3D we used the implementation provided by the authors (KHT-3D...,).

For RANSAC, we chose to compare against the technique of Schnabel et al. (SCHNABEL; WAHL; KLEIN, 2007) and a recent technique by Li et al. (LI et al., 2017). Schnabel et al.'s technique is an efficient RANSAC method for shape detection. The technique of Li et al. (LI et al., 2017) 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. (FARID, 2015), Pham et al. (PHAM et al., 2016), and Vo et al. (VO et al., 2015). 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 3.1.3, 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 work 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 4.6: (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 (LIMBERGER; OLIVEIRA, 2015). Datasets (vi) to (vii) are from Leica's public sample datasets (LEICA...,). 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. The value of our parameters (MDP threshold, MDN threshold and the octree min num of samples) were chosen experimentally, selecting one particular dataset and fine-tuning it until the visual results became good enough. Once they were set, they were kept for all datasets. For the other techniques, however, we adjusted each one for each dataset individually, following a similar procedure where we stopped when we obtained the best visual results after a determined number of tries.

Table 4.1 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 4.1 – Information about each plane detection dataset

	# 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

Source: the author.

4.9.1 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 our system graphical interface. Each plane was obtained using the median of its sample positions, as well as the median of its sample normals. Figure 4.6 (right) displays the obtained ground truth for all datasets, with each plane in a given dataset shown in a different color.

4.9.2 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*.

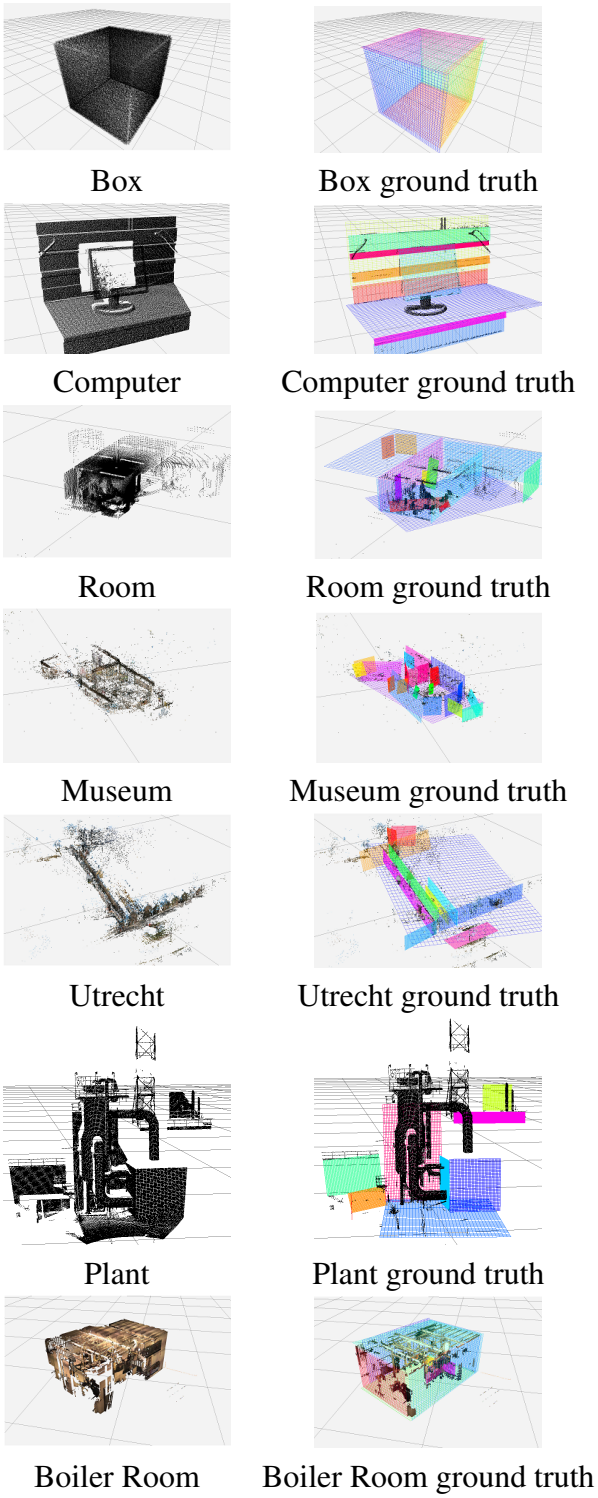
A reliable metric should not just count the number of correctly detected planes, as this may be misleading. Some planes might be more important than others for a given scene. For instance, detecting a large plane should be more important than detecting a small one, since large planes tend to contribute 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 4.7 (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 4.7 (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 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 4.2. They were obtained experimentally to reduce the proximity-bias effect on each dataset and were used to evaluate the results of all techniques.

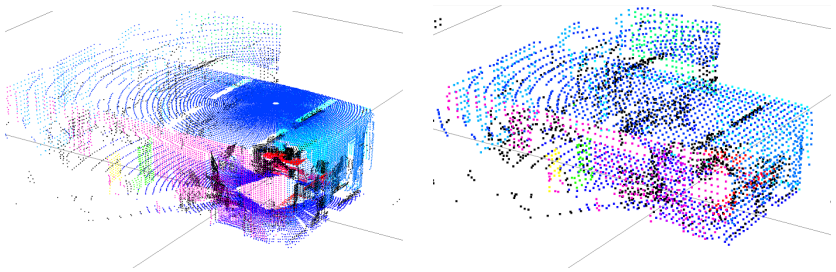
Given the samples of the original point cloud associated to a given plane π_j (either from the ground truth or detected by a technique), if a voxel v_i in the regularized space contains at least one sample belonging to π_j, that voxel is said to be part of the plane (*i.e.*,

Figure 4.6 – Datasets listed in Table 4.1. (left) Point clouds. (right) Corresponding ground truths. Each plane is shown in a different color.



Source: the author.

Figure 4.7 – 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).



Source: the author.

$v_i \in \pi_j$). Note that v_i may be part of multiple planes simultaneously.

A plane π_{GT_k} from the ground truth is considered to be *correctly detected* by a technique T if a plane π_{T_k} in T's list of detected planes satisfies the following conditions: (i) at least 50% of the voxels in π_{T_k} are also in π_{GT_k}; and (ii) the angular difference between the normals of π_{T_k} and π_{GT_k} does not exceed $30°$. The first criterion tries to enforce that π_{T_k} is centered around the same spatial location as π_{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 π_{T_k} if it belongs to both π_{T_k} and π_{GT_k} (*i.e.*, $v_i \in \pi_{T_k}$ and $v_i \in \pi_{GT_k}$), and π_{GT_k} was correctly detected by T. It is a **false positive** with respect to π_{T_k} if $v_i \in \pi_{T_k}$ but $v_i \notin \pi_{GT_k}$. It is a **true negative** with respect to π_{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 π_{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:

$$\begin{aligned} precision_{T_d} &= \frac{(TP)_{T_d}}{(TP)_{T_d} + (FP)_{T_d}}, \\ recall_{T_d} &= \frac{(TP)_{T_d}}{(TP)_{T_d} + (FN)_{T_d}}, \\ F1_{T_d} &= 2 \times \frac{precision_T \times recall_T}{precision_T + recall_T}, \end{aligned} \tag{4.12}$$

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 4.2 – 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

Source: the author.

4.9.3 Time and Quality Analysis

We ran the experiments comparing the performances of the various techniques on an Intel Core™ i7-7700K 4.20GHz CPU with 32GB RAM. Table 4.3 summarizes the results for each dataset.

To prevent bias, all techniques that require normals (RANSAC (Schnabel) (SCHNABEL; WAHL; KLEIN, 2007), RG (Farid) (FARID, 2015), and RSPD (Ours)) used the same sample normals obtained with MCD (ROUSSEEUW; DRIESSEN, 1999), as described in Section 2.1.3. The same neighborhoods were used to create the neighborhood graphs used by our technique, RG (Farid) and RG (Vo) (VO et al., 2015).

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 4.3 summarizes the performance of the compared plane detection techniques on the seven datasets listed in Table 4.1. 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 4.8.

Table 4.3 – 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

Source: the author.

Table 4.4 – 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

Source: the author.

4.10 Discussion

With the exception of RANSAC (NDT) by Li et at. (LI et al., 2017), all the techniques performed well on the Box dataset (Figure 4.8, 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, 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.

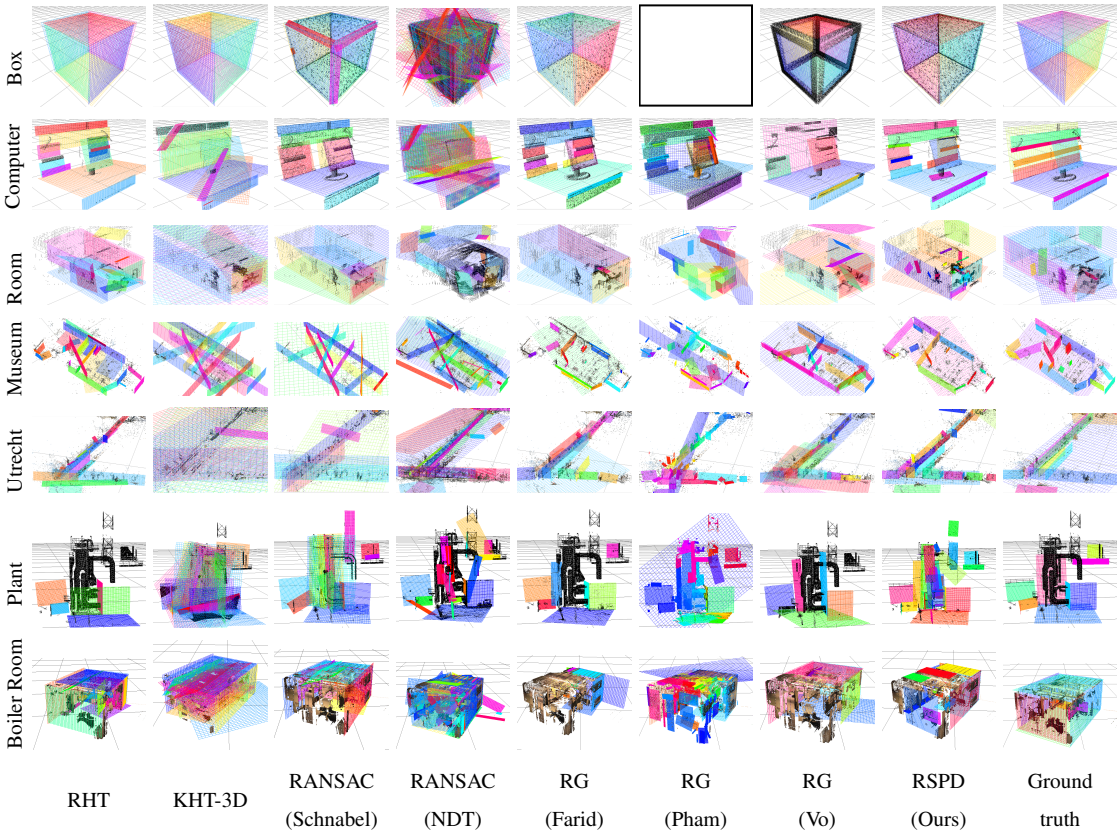
The Museum dataset contains the largest number of planes (23), most of them small. Therefore, techniques able to analyze connected components and delimit planar regions performed best. Our technique was able to retrieve constrained planes in half of a second. KHT-3D, RANSAC (Schanabel), and RANSAC (NDT) 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

Figure 4.8 – 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. We could not find a set of parameters to execute RG (Pham) on the Box dataset in reasonable time.



Source: the author.

second best overall precision, while being the one of the fastest techniques (Table 4.4). In terms of time, RG(Vo) and KHT-3D performed extremely well in all datasets, having similar running times.

5 CYLINDER DETECTION

This chapter discusses our automatic cylinder-detection technique. In industrial sites, cylinders represent pipes, tanks, and ducts, which along with planes are the most common structures found in these environments.

5.1 Overview

In order to detect cylinders with arbitrary orientations, our technique projects the point cloud onto a set of uniformly-distributed directions on a unit hemisphere (Section 5.2). Each direction defines a tangent plane onto which we orthographically project the samples whose normals are approximately perpendicular to the plane normal (Figure 5.1).

We then refine the orientations of these projection planes and re-project the samples onto them (Section 5.4). For this, for each group g_i of projected samples that form a connected component in 3D, we compute a new plane orientation applying PCA to the normals of these samples in 3D. Such samples are then re-projected onto the new plane. Then, a novel circle-recognition technique is applied to elements of g_i to detect circular projections (Section 5.5).

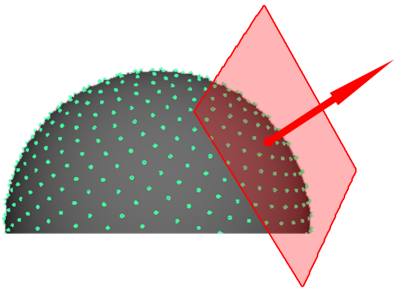
Delimited cylindrical surfaces are finally obtained by merging related connected components in 3D and fitting cylinders to the merged components (Sections 5.7 to 5.9). Samples belonging to detected cylinders are removed from the point cloud, and the remaining connected components are evaluated. This process is illustrated in Figure 5.2 and on Algorithm 4. Its details are presented in Sections 5.2 to 5.9.

5.2 Projection directions

Given a circular cylinder, the projection of its samples onto a plane perpendicular to the cylinder's axis defines a circle. Thus, finding circular patterns resulting from projected samples can be used to greatly simplify the detection of cylinders in point clouds.

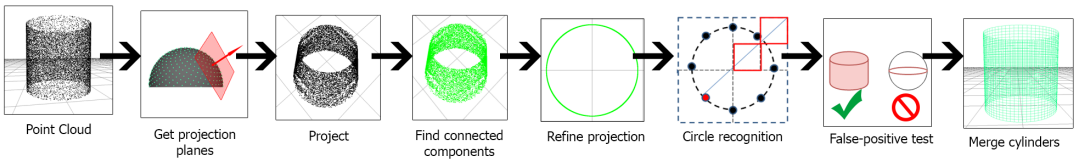
Unfortunately, plane detection techniques applied on the Gauss map and used for cylinder detection (LIU et al., 2013; CHAPERON; GOULETTE, 2001; RABBANI; HEUVEL, 2005) are error prone and computationally expensive. Thus, in order to select

Figure 5.1 – Uniform sampling of a hemisphere defining the initial projection orientations.



Source: the author.

Figure 5.2 – Our automatic cylinder detection pipeline. Given an input point cloud, it is projected along a set of directions on the unit hemisphere. These directions are further refined. Projected circles are detected and outliers removed. Cylindrical surfaces are then obtained by fitting cylinders to connected components in 3D corresponding to detected circles. The samples of detected cylinders are removed from the point cloud, and related components are merged into single cylinders.



Source: the author.

Algorithm 4 Cylinder Detection

Require: PC {point cloud with estimated normals and neighborhood information} N_s {number of sampling directions}

procedure DETECTCYLINDERS(PC, N_s)
  $cylinders \leftarrow \emptyset$
  $components \leftarrow \emptyset$
  $directions \leftarrow Fibonacci(N_s)$ ▷ sampling directions: Fibonacci mapping
  for all $D \in directions$ **do**
    $P' \leftarrow Project(PC, D)$
    $c \leftarrow FindConnectedComponents(P')$
    $components.insert(c)$
  for all $c \in components$ **do**
    $P' \leftarrow Reproject(c)$
    if $ContainsCircle(P')$ **then**
      $Cylinder \leftarrow FitCylinder(c)$
      if $IsValid(Cylinder)$ **then**
        $cylinders.insert(Cylinder)$
        $RemoveSamples(c)$
        $components \leftarrow FindNewConnectedComponents(components)$
  return $cylinders$

the projection directions, we perform an initial uniform sampling of the unit hemisphere, which is further refined to adjust them to the point cloud content. Any uniform sampling technique can be used, but we opt for the spherical Fibonacci mapping (KEINERT et al., 2015) (due to its simplicity), with 100 sampling directions (Figure 5.1). Algorithm 5 implements the Fibonacci mapping.

5.3 Detecting connected components

Given the projection directions defined by the Fibonacci mapping, we project the point cloud along each direction $d_{(\theta,\phi)}$, considering only the samples whose normals are approximately perpendicular to $d_{(\theta,\phi)}$ (*i.e.*, $d_{(\theta,\phi)} \pm \tau$). For all results shown in the paper, we used an angular tolerance $\tau = 10°$.

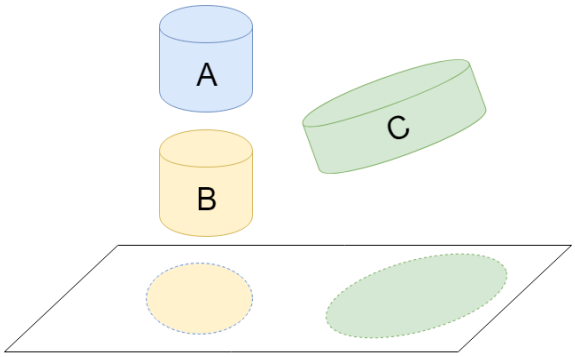
Naively detecting circles on each projection can be, not only computationally expensive, but also error prone, for two reasons: (i) real scenes may contain cylinders whose projections may exactly overlap, causing two (or more) cylinders to be detected as a single one; and (ii) none of the projection planes may be perpendicular to a given cylinder axis, resulting in projected ellipses, which may not be detectable by standard approaches. Both situations are illustrated in Figure 5.3.

Algorithm 5 Hemispherical Fibonacci Mapping

Require: N_s {number of sampling directions}

procedure FIBONACCI(N_s)
  $directions \leftarrow \emptyset$
  $offset \leftarrow 2/N_s$
  $increment \leftarrow \pi(3 - \sqrt{5})$
  for $i \leftarrow 1$ to $2N_s$ **do**
    $y \leftarrow i \times offset + offset/2 - 1$
    $r \leftarrow \sqrt{1 - y^2}$
    $\alpha \leftarrow Modulo(i + 1, N_s) \times increment$
    $x \leftarrow rcos(\alpha)$
    $z \leftarrow rsin(\alpha)$
    if $z > 0$ **then**
      $directions.insert([x, y, z])$
  return $directions$

Figure 5.3 – The projections of cylinders A and B overlap. The projection of cylinder C produces an ellipse.



Source: the author.

To avoid having two (or more) cylinders detected as a single one, we split each projection into groups of samples (g_i's), where each group forms a connected component in 3D. To obtain the connected components, we compute a neighborhood graph G as described in Section 2.1.2, with a neighborhood of size $k = 50$. Then, for each projected sample, we perform a breadth-first search (BFS) on G considering *only* the set of projected samples. This search will naturally return a connected component. We perform searches until all projected samples have been visited.

5.4 Refine Projection Plane Orientations and Samples

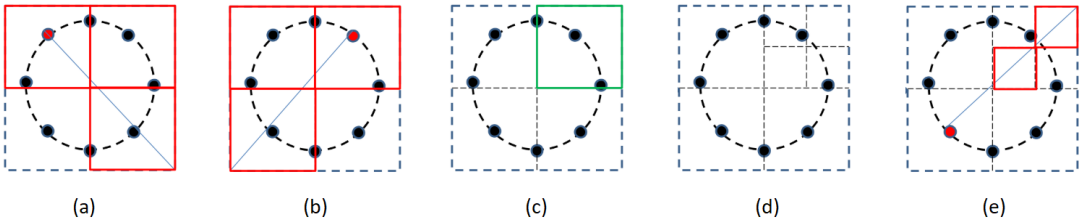
In order to avoid elliptical projections, the projection directions need to be refined for some cylinders. These will correspond to the new cylinder axes. Thus, let c_i be a connected component in 3D belonging to a cylinder $\mathcal{C}_j$ in the point cloud, and corresponding to the projected samples in g_i. Since c_i's samples should have normals perpendicular to $\mathcal{C}_j$'s axis, we estimate the cylinder axis by applying principal component analysis (PCA) to the set of normals of c_i's samples. The direction with least variance corresponds to $\mathcal{C}_j$'s axis. We can then project c_i's samples onto the plane perpendicular to newly estimated $\mathcal{C}_j$'s axis using the procedure described in Section 5.3. Both cylinder axis refinement and sample reprojection are shown in Algorithm 6.

Algorithm 6 Refine cylinder axis and samples

Require: c {connected component} G {neighborhood graph}

procedure READJUSTCOMPONENT(c, G)
  $pca \leftarrow PCA(c.normals)$
  $axis \leftarrow pca[1]$ ▷ eigenvector with smallest eigenvalue: $|\lambda_1| \leq |\lambda_2| \leq |\lambda_3|$
  $q \leftarrow \emptyset$ ▷ initialize sample queue
  for all $s \in c.samples$ **do**
    $q.enqueue(s)$
  while $q \neq \emptyset$ **do**
    $front \leftarrow q.dequeue()$
    for all $n \in G.neighbors(front)$ **do**
      if $n \notin c \wedge acos(dot(n.normal, axis)) > \alpha$ **then**
        $c.insert(n)$
  return $Project(c, axis)$

Figure 5.4 – The subdivision strategy used to find a circle. At first, rays are traced from each sample position in direction to its normal (a) (b), then the quadtree cell with most intersections is chosen (c) and subdvided (d), where the process is performed recursively (e), until the quadtree cell size becomes small enough.



Source: the author.

5.5 Circle Recognition

Once each connected component has been refined (Sections 5.3 and 5.4), our technique checks if its projection fits a circle (see pipeline in Figure 5.2). For this, we introduce a fast and robust circle recognition technique by exploring the fact that extended normals from each sample of a circle should intersect at the circle's center. One can then classify a set of samples as being on a circle or not depending on the existence of such *point of intersection*, which can be quickly checked using a subdivision strategy.

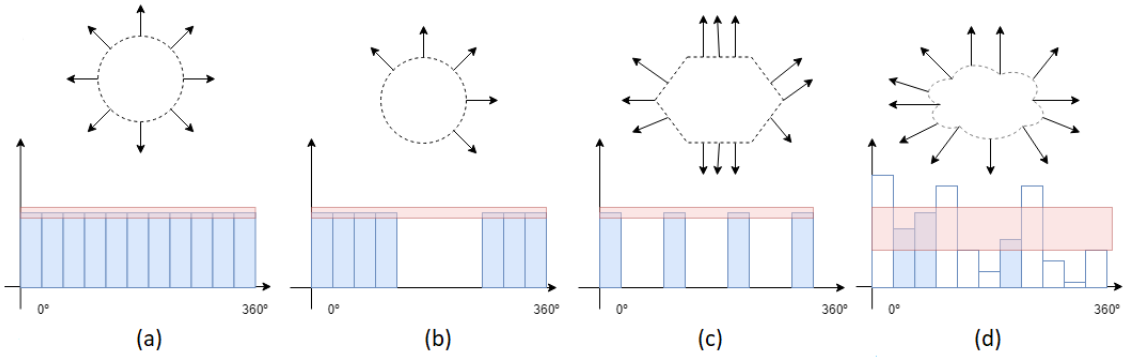
It works as following: consider a quadtree, initialized with the bounding box of the set of projected samples, and containing four child nodes (Figure 5.4 (a)). For each projected sample in g_i, we check if the (reverse) ray formed by its position and (reverse) projected normal intersects the bounding box of each quadtree's child, as shown in Figure 5 (a), in which case the sample is added to the child's list of intersections. Algorithm 7 summarizes this procedure.

Algorithm 7 Check 2D Ray-AABB Intersection

Require: R {Ray} BB {Bounding Box}

procedure INTERSECT(R, BB)
- $tx_1 \leftarrow (BB.x_{left} - R.O_x)/R.N_x$ ▷ $R.O = Origin$
- $tx_2 \leftarrow (BB.x_{right} - R.O_x)/R.N_x$ ▷ $R.N = Direction(normalized)$
- $t_{min} = min(tx_1, tx_2)$
- $t_{max} = max(tx_1, tx_2)$
- $ty_1 \leftarrow (BB.y_{up} - R.O_y)/R.N_y$
- $ty_2 \leftarrow (BB.y_{down} - R.O_y)/R.N_y$
- $t_{min} = max(t_{min}, min(ty_1, ty_2))$
- $t_{max} = min(t_{max}, max(ty_1, ty_2))$
- **return** $t_{max} \geq t_{min}$

Figure 5.5 – Histogram for different shapes after mapping the range of the *arctan2* function from $[-\pi, \pi)$ to $[0°, 360°)$. Circular shapes have their bins more uniformly distributed (a) and (b), while the bins of other shapes are either too sparse (c), or non-uniformly distributed (d).



Source: the author.

After checking intersections for all samples in g_i, the quadtree child node with most intersections is chosen and checked to determine if the projected samples whose (reverse) rays intersect it form a circular shape (Figure 5.4 (b)). This is done by first computing a histogram of the angles measured between the projected normal directions and the horizontal axis, according to Equation 5.1. If a shape is circular, this histogram must be uniformly distributed (Figure 5.5 (a)). However, this would only happen in case the shape is a full circle. Since point clouds are susceptible to occlusion, it is desirable to also consider arcs of varying lengths. Thus, we check if a percentage of the angular bins are uniformly distributed (*i.e.*, if the number of elements in each bin is approximately equal to the ratio between the number of projected samples and the number of bins). This is illustrated in Figure 5.5 (b). Figures 5.5 (c) and (d) show examples of histograms associated with non-circular projections.

$$\theta_N = arctan2(N_y, N_x) \tag{5.1}$$

A bin element is considered part of a uniform bin distribution if its value falls in the interval built around the mean (*i.e.*, number of projected samples in g_i divided by the number of bins), μ_{bins}, using three standard deviations of the bin values, σ_{bins}: $[\mu_{bins} - 3\sigma_{bins}, \mu_{bins} + 3\sigma_{bins}]$. For the examples shown in this work, we subdivided the histogram into 72 bins, and considered the minimum acceptable number of uniformly-distributed bins to be 18 (one quarter of the total number of bins, or equivalent to 90 degrees). If the quadtree child node with most intersections does not meet the above con-

dition, the connected component is immediately rejected as a circle candidate. Otherwise, we subdivide the child node and call the same procedure recursively up to five times, considering only the set of samples in g_i whose associated reverse rays intersect the child node. This process is illustrated in Figure 5.4 (c) and (d).

Algorithm 8 Find a circle in a set of samples

Require: Q {Quadtree} h Minimum quadtree height

1: **procedure** FINDCIRCLE(Q, h)
2: **if** $Q.height > h$ **then**
3: $FilterNoise(Q_{samples})$
4: **return** $Q_{samples}$
5: **if** $IsNotCircular(Q)$ **then**
6: **return** $NULL$
7: **for all** $p \in Q_{samples}$ **do**
8: **for all** $q \in Q_{children}$ **do**
9: **if** $Intersect(p, q)$ **then**
10: $q_{samples}.insert(p)$
▷ Q's child node with maximum number of intersections
11: $q_{withMaxInters} \leftarrow MaxIntersection(Q_{children})$
12: **return** $FindCircle(q_{withMaxInters}, h)$

The main advantage of our circle-detection algorithm over classical approaches, such as Hough transform and RANSAC, is its lower computational cost. Being significantly faster than previous techniques, our solution enables testing the projection of the point cloud along a larger number of directions in the same amount of time, ultimately improving the accuracy of cylinder detection. Although there are existing circle detection techniques which are faster than classical approaches, they are mostly targeted to rastered images, and not really applicable to the unorganized point cloud domain, and thus the need for a new technique.

Given N samples, the computational cost of our circle-detection technique is $O(D \times N)$, where D is the maximum depth of the quadtree. Since D is small ($D = 5$ for all examples shown in the paper), the cost is $O(N)$. The Hough transform for circle detection has cost $O(bins_x \times bins_y \times bins_{radius} \times N)$, where $bins_x$, $bins_y$, and $bins_{radius}$ are the number of bins in each dimension of the accumulator. RANSAC, in turn, has cost $O(I \times N)$, where I is the number of iterations required to detect a circle, which can be arbitrarily high, depending on the noise level and the specified thresholds.

Another advantage of our technique is its independence of noise-level thresholds. For RANSAC, in particular, it is difficult to set a good distance threshold which works well for all projected samples from a point cloud. This happens because such threshold

depends on the noise level, which is not uniform in most unorganized point clouds (the further a sample is from the sensor, the stronger the noise level).

5.6 Robust Outlier Removal

Once a circle has been detected by the technique described in Section 5.5, we perform an outlier-removal procedure before fitting a cylinder to the set of samples whose projections were used to detect the circle. For each g_i, its samples are used to obtain robust estimates for a circle center and radius.

To estimate the circle center, we take n groups of three projected samples (we set n equal to the number of samples in g_i) and from the j-th triple we estimate a circle and its center at (C_x^j, C_y^j), creating two new sets of observations: $C_{ix} = [C_x^1, C_x^2, ..., C_x^n]$ and $C_{iy} = [C_y^1, C_y^2, ..., C_y^n]$. The circle center is estimated as the median of each set, *i.e.*, $MC_i = (median(C_{ix}), median(C_{iy}))$.

The radius MR_i of the circle is obtained as the median of the set $\Delta_i = \{\delta_{i1}, \delta_{i2}, ..., \delta_{in}\}$ of distances from the projection of each sample $s_j^{g_i} \in g_i$ to MC_i. In order to robustly detect outliers with 99.7% of confidence, we define an interval I_i centered at MR_i with 3 $MADs$ of extent to each side (Equation 5.2). Any sample whose distance to $MC_i = median(\Delta_i)$ falls outside this interval is discarded as an outlier. This process is illustrated in Figure 5.6.

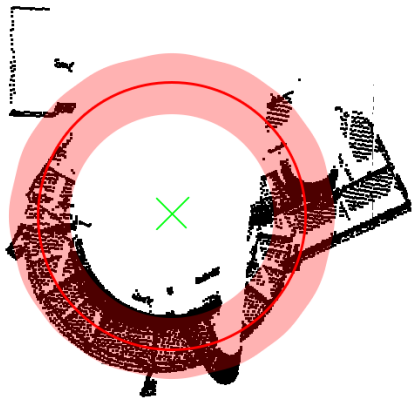
$$I_i = [median(\Delta_i) - 3 \times MAD(\Delta_i); median(\Delta_i) + 3 \times MAD(\Delta_i)]. \quad (5.2)$$

5.7 Detecting False Positives

The detection of a projected circle does not guarantee that the corresponding connected component forms a cylinder. Projections of other 3D shapes, such as spheres and cones, also produces circles (Figure 5.7). Thus, a mechanism for detecting false positives is required.

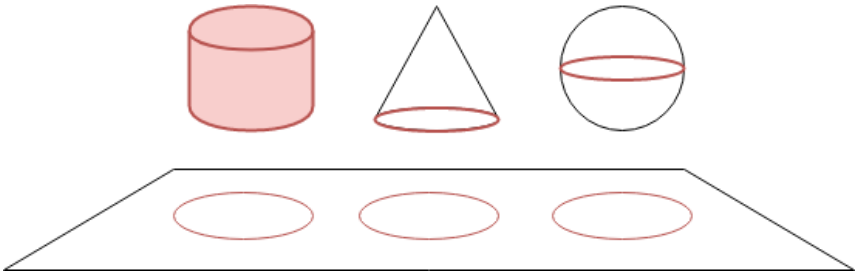
After removing outliers from g_i, our technique uses a least-squares procedure to fit a cylindrical surface to the set of corresponding samples in the associated connected component c_i in 3D. It is based on Equation 5.3, which computes the distance from a

Figure 5.6 – Outlier removal. Red circle obtained using robust estimates for its center (green X, MC_i) and radius MR_i. The pink disk represents the interval defined by Equation 5.2. Samples (black dots) outside this interval are discarded as outliers.



Source: the author.

Figure 5.7 – Samples from objects with different shapes may produce circular projections.



Source: the author.

given sample $s_j \in c_i$ to the cylindrical surface:

$$d_j = \left(\frac{||(s_j - \mathcal{C}_c) \times (s_j - (\mathcal{C}_c + A))||}{||(\mathcal{C}_c + A) - \mathcal{C}_c||} - r \right)^2 = (||(s_j - \mathcal{C}_c) \times (s_j - \mathcal{C}_c - A)|| - r)^2 , \tag{5.3}$$

where s_j is a sample position in 3D, $\mathcal{C}_c$ is the cylinder center, A is the cylinder axis and r is the cylinder radius. And since the cross product is a linear transformation,

$$\begin{aligned} d_j &= (||(s_j - \mathcal{C}_c) \times (s_j - \mathcal{C}_c) - (s_j - \mathcal{C}_c) \times A|| - r)^2 \\ &= (||(\mathcal{C}_c - s_j) \times A|| - r)^2. \end{aligned} \tag{5.4}$$

After the fitting, a cylinder is considered valid if it meets three conditions: (i) at

least 50% of the normals from the samples used to fit the cylinder are perpendicular to the fitted cylinder axis (with tolerance $\tau = 10°$); (ii) at least one quarter of the bins of the angular histogram (*i.e.*, at least $90°$) are uniformly distributed; (iii) the ratio between the fitted cylinder radius and its height must be below a threshold γ (conservatively set to 5). Conditions (i) and (ii) evaluate the quality of the fitting. Condition (iii) prevents cones, spheres, and related geometric shapes from being detected as cylinders (Figure 5.7). For a non-cylindrical geometric shape whose projection produces a circle, only a small section of it will be actually projected. Such a section can be bigger or smaller depending on the angular threshold τ allowed between the sample and plane normals. Regardless, it is expected that the height/radius ratio for false positive cylinders be small and much smaller than $\gamma = 5$.

5.8 Recomputing Connected Components

The samples associated with detected cylinders are removed from the point cloud. This may affect connected components that may not have been analyzed yet, as a sample may belong to more than one connected component (*e.g.*, consider a sample at the intersection of two cylinders).

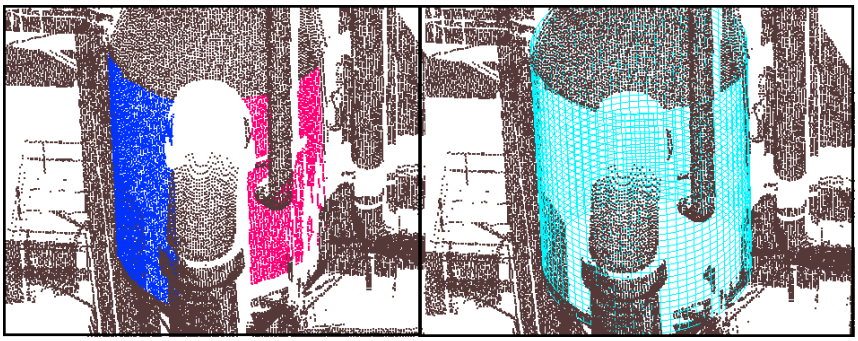
Thus, after removing samples, the connectivity of all components that have not been evaluated yet need to be re-checked. If a connected component has been split, the original component is removed and its subcomponents are added to the list of connected components.

In order to speed up this process, the largest components are analyzed first, as we want to remove the largest number of samples as soon as possible, preventing them from being projected and analyzed multiple times unnecessarily. After recalculating the connected components, they are sorted based on their number of samples.

5.9 Merging Connected Components Belonging to the Same Cylinders

A cylinder may be fragmented into multiple connected components. Since our technique treats each connected component separately, a cylinder may be detected multiple times, once per fragment, and these components need to be merged. This situation is illustrated in Figure 5.8.

Figure 5.8 – Merging multiple connected components belonging to a cylinder. (left) Due to partial occlusion, two connected components from the same cylinder in the Petrochemical plant dataset, marked in blue and red, are detected as possibly belonging to separate cylinders. (right) The resulting detected cylinder after the merging process.



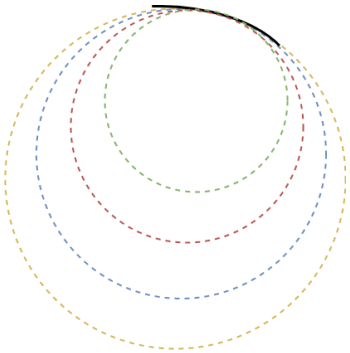
Source: the author.

To identify the cylinders whose components should be merged, we iterate over each pair of detected cylinders $\mathcal{C}_i$ and $\mathcal{C}_j$, which should satisfy three similarity tests: (i) they must have similar orientations (*i.e.*, similar axis directions); (ii) they must have similar radii values; and (iii) they must have similar center positions. The connected components that satisfy such criteria are merged using union-find operations. After all merge operations have been performed, a resulting cylinder is obtained from each union through least-squares fitting of its samples (Figure 5.8 (right)).

Test (i) checks if the angle between the axes of $\mathcal{C}_i$ and $\mathcal{C}_j$ is below a certain threshold α (experimentally set to $10°$). Test (ii) checks if the ratio $max(r_i, r_j)/min(r_i, r_j)$ between the radii r_i and r_j of the two cylinders is below a certain threshold β (experimentally set to 2). For test (iii), let l_i and l_j be the line segments corresponding to the limits of the projections of the samples in the connected components c_i and c_j on the axes of $\mathcal{C}_i$ and $\mathcal{C}_j$, respectively (Figure 5.10). l_i and l_j approximate the medial axes of the surfaces defined by c_i and c_j, respectively. For the connected components of $\mathcal{C}_i$ and $\mathcal{C}_j$ to be merged, the distance between l_i and l_j should be below a certain threshold (experimentally set as $max(r_i, r_j)/10$).

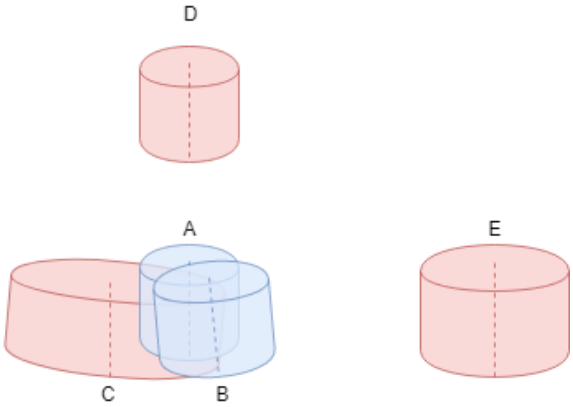
We opted for a large threshold β for the cylinders' radii ratio because the radius value estimated by least squares is not as reliable as the estimated axis, especially when just a small section of a cylinder is available (*i.e.*, a $90°$ arch). This situation is illustrated in Figure 5.9.

Figure 5.9 – Radius estimation uncertainty. Four cylinders (top view) with different radii estimated from the samples shown in black (solid arch). Although the differences in radii are big, the actual fitting errors are small in all four cases.



Source: the author.

Figure 5.10 – Evaluating multiple conditions to merge the connected components from two cylinders. Cylinders A and B will be merged because they have similar orientations, radii, and distance from their medial axes (dotted line segments) is sufficiently small. Cylinder C will not be merged with A or B because its radius is much bigger than the other two. Cylinder D, despite having same orientation and radius, will not be merged with cylinder A because the distance between their medial axes is too big. The same applies to cylinder E with respect to cylinders A and B.



Source: the author.

5.10 Complexity Analysis

The cost of our automatic cylinder detection technique is $O(pn^3)$, where p is the number of projection directions on the uniform sampling of the hemisphere (Figure 5.1), and n is the number of samples in the point cloud. Next, we analyze the cost of the individual steps of the algorithm.

For each projection direction, we analyze each sample and project it if its normal is approximately perpendicular to the given direction. This has cost $O(pn)$. For each projection, we compute the connected components using a neighborhood graph G. We compute G using k-nearest neighbors, which for an individual sample has cost $O(klog(n)) = O(log(n))$ (using a kd-tree as auxiliary data-structure, where k is the number of neighbors, and assuming $k << n$). For all n samples, the cost of computing G is $O(nlog(n))$.

Since we perform a breadth-first search on G to compute the connected components, this step has cost $O(e + n)$, where e is the number of edges (neighbors) and n is the number of vertices (samples) in G. Since, for G, $e << n$, this has cost $O(n)$.

The projection directions are refined, with more samples being included in the connected component, if necessary. Such refinement is performed only once using PCA (ZOU; HASTIE; TIBSHIRANI, 2006), whose cost is $O(n^3)$. Thus, this whole refinement process has cost $O(pn^3)$.

For each connected component, we perform a circle recognition using a quadtree to check the intersection of the sample normals with the quadtree node cells (Algorithm 7). For each level, only one cell is subdivided, and in the worst case this cell will be intersected by rays from all samples. This has cost $O(dn)$, where d is the maximum number of quadtree subdivisions for a circle to be recognized. Therefore, this step has cost $O(pdn)$. However, since $d \leq 5$, this step has cost $O(pn)$.

For the robust outlier removal, we calculate mean and MAD of a set of samples to obtain the inlier interval. Calculating the median of a set of distances can be performed in $O(n)$, and thus this step has cost $O(n)$.

In the false-positive test, the most expensive operation is the least-squares fitting of a set of samples, whose cost is $O(n^3)$. Since one sample can be projected at most p times, this test has cost $O(pn^3)$.

For merging connected components from multiple cylinders, each cylinder is compared to all the others. Since the minimal theoretical number of samples required to rep-

resent a cylinder is five (CHAPERON; GOULETTE, 2001), in the worst case we have $n/5$ cylinders. Thus, this step has cost $O(n^2)$. Therefore, the total cost of our technique is $O(pn^3)$.

5.11 Results

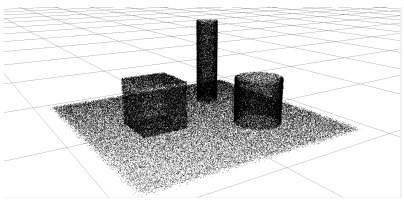
In order to evaluate our technique, we used five datasets consisting of two synthetic (including a complex one that models an oil refinery) and three obtained from real scenes (Figure 5.11). Such models were chosen to stress the ability of cylinder-detection techniques to handle different features and configurations found in actual installations. The datasets are: (i) *Synthetic scene*: a scene containing one cube and two cylinders on top of a plane. The positions of these samples were corrupted using a uniform distribution of noise values ranging from 0 to 1% of the side of the point cloud's cubic bounding box. (ii) *Synthetic oil refinery*: an oil refinery scene modeled using 3D software and converted to point cloud by uniformly sampling the polygonal mesh; (iii) *Pump room*: a sewer treatment plant pump room; (iv) *Petrochemical plant*: a frontal scan from a petrochemical site; and (v) *Boiler room*: a set of integrated scans from the complex environment of a boiler room consisting of almost six million samples. The real datasets were obtained from Leica's public sample repository (LEICA...,). The ground truth for each dataset was obtained by manually selecting the samples of each cylinder and using them to least-squares fit a cylindrical surface. Table 5.1 shows the number of cylinders and the percentage of the scene area covered by cylindrical surfaces in each dataset.

Table 5.1 – Dataset Information: Number of Cylinders and Percentage of Cylindrical Scene Coverage

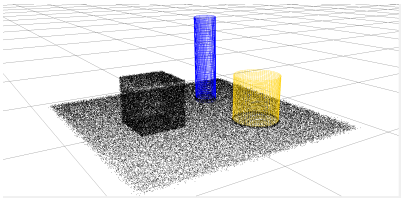
	#samples	#cylinders	% cylinder scene coverage
Synthetic scene	138,000	2	34
Synthetic oil refinery	499,976	3	22
Pump room	166,976	13	13
Petrochemical plant	358,116	8	13
Boiler room	5,990,481	21	7

Source: the author.

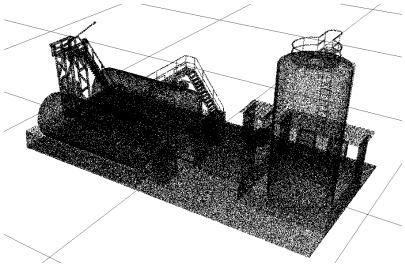
Figure 5.11 – Ground-truth for each dataset.



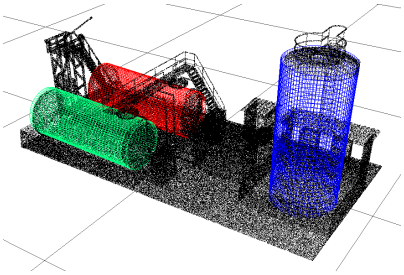
Synthetic scene



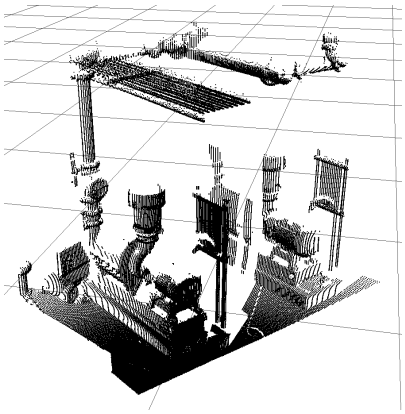
Synthetic scene ground truth



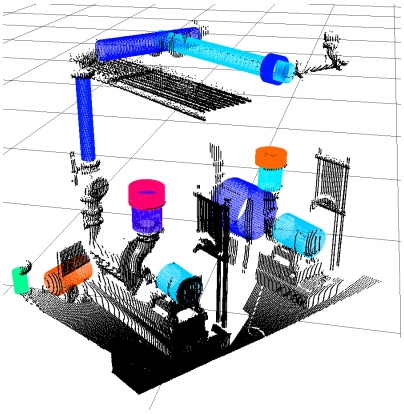
Synthetic oil refinery



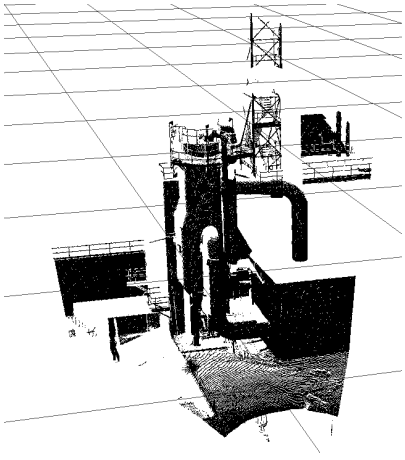
Synthetic oil refinery ground truth



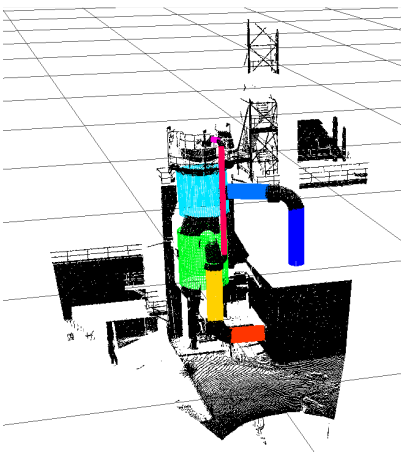
Pump room



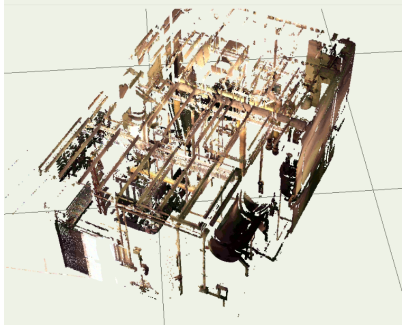
Pump room ground truth



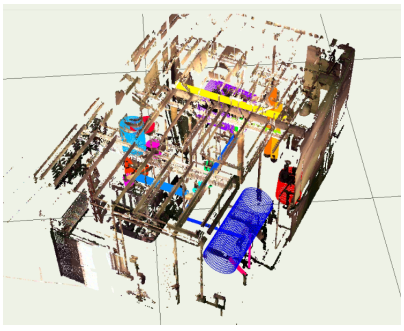
Petrochemical plant



Petrochemical plant ground truth



Boiler room



Boiler room ground truth

Source: the author.

5.11.1 Evaluation metrics

In order to objectively compare our technique to previous ones, we adapted three well-known metrics from Information Retrieval to our context: precision, recall, and F1-score. *Precision* is the percentage of correctly retrieved instances (*i.e.*, true positive) among all retrieved ones (*i.e.*, true positive + false positive). *Recall* is the percentage of correctly retrieved instances among all correct instances (*i.e.*, true positive + false negative). Finally, *F-1 Score* is the harmonic mean of precision and recall. These measures are summarized by Equations 5.5 to 5.7.

$$Precision = \frac{true\ positive}{true\ positive + false\ positive} \tag{5.5}$$

$$Recall = \frac{true\ positive}{true\ positive + false\ negative} \tag{5.6}$$

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{5.7}$$

We say that a detected cylinder corresponds to a true positive if its orientation differs by no more than 20° from the ground truth's orientation, and they share at least 50% of their samples. The first condition ensures that the cylinders have similar orientations, while the second ensures that they are centered at approximately the same position in space.

5.11.2 Experiments

Our technique was implemented in C++ using Eigen(EIGEN...,) as our linear algebra library. We compared it against the most popular as well as the most recent approaches for cylinder detection using the five datasets shown in Figure 5.11. These methods are based on RANSAC, Hough transform, and region growing. All experiments were executed on a Intel® Core™ i7-7700K 4.20GHz CPU with 32GB RAM.

Next, we describe the compared techniques and present the reasons for their choices. In the Hough transform category, the work of Ahmed et al. (AHMED; HAAS; HAAS, 2014) was chosen because it corresponds to a state-of-art technique for cylinder detection. In terms of RANSAC, the work of Schnabel et al. (SCHNABEL; WAHL; KLEIN,

2007) was chosen since it is a popular and traditional RANSAC approach. Liu et al. (LIU et al., 2013) was chosen because it is a recent approach to detect pipes, which is also the main intent of our work. Finally the work of Tran et al. (TRAN; CAO; LAURENDEAU, 2015) was chosen since, to our knowledge, it is the most recent work on cylinder detection. It is based on region growing.

With the exception of Schnabel et al. (SCHNABEL; WAHL; KLEIN, 2007), we could not find implementations of the other techniques. Thus, we implemented them ourselves, as faithfully as possible, also in C++. In order to prevent bias, for techniques that require normals, we used the same normal estimation approach used for our technique. Table 5.2 summarizes the results of our experiments, including values for precision, recall, F1-score, and execution time in seconds. Note that our technique achieved the best F-1 score for all evaluated datasets, while still maintaining a competitive running time. The ground truths and the cylinders detected by each technique are shown in Figure 5.11.

Table 5.2 – Performance of the evaluated techniques on each dataset. Number of detected cylinders over total number of cylinders (#), Precision (P), Recall (R), F1-Score (F1), Elapsed time in seconds (T(s)). Best results highlighted in bold.

	#	P	R	F1	T(s)
Synthetic scene					
RANSAC (Schnabel)	2/2	0.41	**0.94**	0.58	0.26
RANSAC (Liu et al.)	1/2	**0.99**	0.41	0.58	0.37
RG (Tran et al.)	2/2	0.91	0.33	0.49	9.64
Hough (Ahmed et al.)	2/2	1.0	0.29	0.45	10.01
Our technique	2/2	0.98	0.89	**0.93**	1.43
Synthetic oil refinery					
RANSAC (Schnabel)	3/3	0.43	**0.93**	0.59	0.22
RANSAC (Liu et al.)	3/3	**0.99**	0.33	0.50	1.75
RG (Tran et al.)	3/3	0.86	0.57	0.68	171.46
Hough (Ahmed et al.)	3/3	**0.99**	0.35	0.52	102.26
Our technique	3/3	0.78	0.63	**0.70**	5.48
Pump room					
RANSAC (Schnabel)	2/13	0.08	0.33	0.14	0.16
RANSAC (Liu et al.)	1/13	0.17	0.08	0.11	0.65
RG (Tran et al.)	3/13	0.65	0.31	0.42	21.38
Hough (Ahmed et al.)	1/13	0.39	0.12	0.19	1.41
Our technique	3/13	**0.78**	**0.36**	**0.49**	0.60
Petrochemical plant					
RANSAC (Schnabel)	6/8	0.14	**0.70**	0.24	0.26
RANSAC (Liu et al.)	4/8	0.27	0.31	0.29	1.03
RG (Tran et al.)	3/8	0.85	0.30	0.45	27.51
Hough (Ahmed et al.)	5/8	**0.86**	0.42	0.56	5.04
Our technique	5/8	0.60	0.63	**0.62**	2.23
Boiler room					
RANSAC (Schnabel)	1/21	0.04	0.23	0.07	5.75
RANSAC (Liu et al.)	1/21	0.02	0.08	0.03	30.13
RG (Tran et al.)	2/21	0.50	0.13	0.20	1159.73
Hough (Ahmed et al.)	0/21	0	0	-	134.58
Our technique	12/21	**0.74**	**0.55**	**0.63**	36.16

Source: the author.

5.12 Discussion

For the *Synthetic scene* dataset, our technique achieved the best results. RANSAC-based techniques such as Schnabel et al.'s tend to detect planar surfaces as spurious incomplete cylinders with very large radius. Thus, for this dataset, Schnabel et al.'s detected the base plane as a spurious cylinder. Liu et al.'s was unable to detect one of the cylinders. Ahmed et al.'s detected the cylinders, but with some missing slices. Tran et al.'s detected spurious cylinders in regions with high curvature, such as the edges of the cube.

For the *Synthetic oil refinery* dataset, our technique was able to detect all the cylinders. However, it overextended the horizontal cylinders, as some close-by samples from the ground plane, whose normals are perpendicular to the axes of these cylinder, were mistaken as belonging to these cylinders. Schnabel et al.'s again detected planes as spurious cylinders. Liu et al.'s was able to obtain a clean detection of the two horizontal cylinders. However, for the vertical cylinder, the circle detection did not obtain a good fit, reducing its recall. Ahmed et al.'s detected all cylinders but for the horizontal ones it only detected small sections of them. Tran et al.'s detected spurious cylinders on the edges.

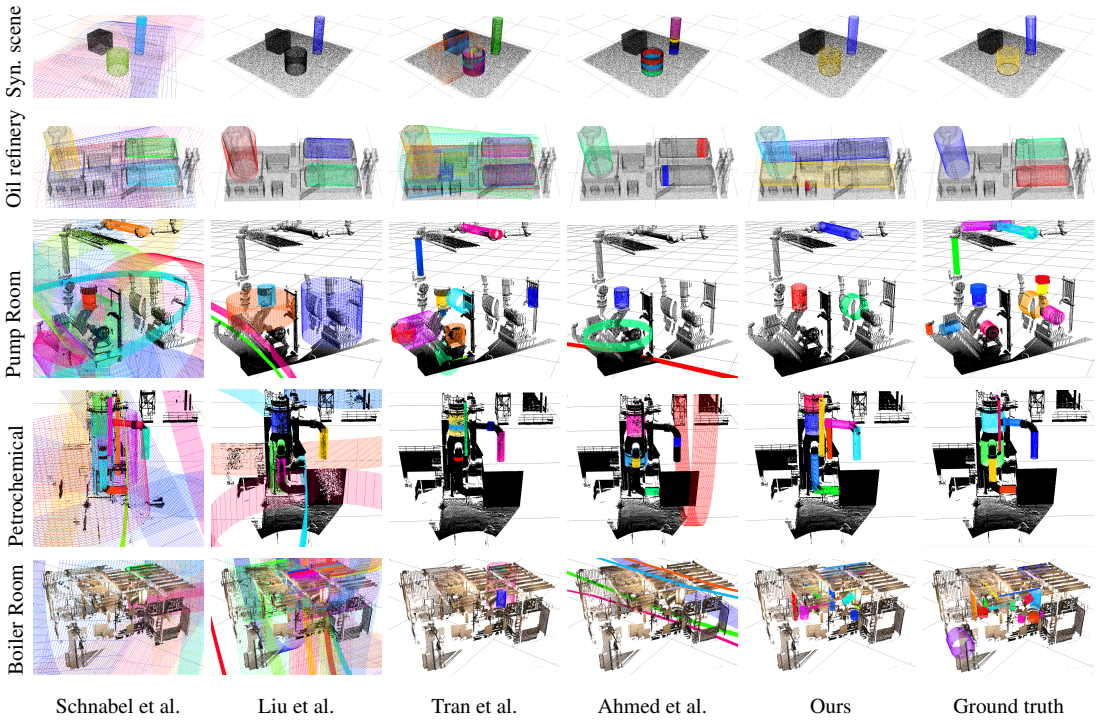
The *Pump room* dataset proved to be quite hard because it only contains partial views of all cylinders. Despite only detecting 3 of the 13 cylinders in the scene, our technique still obtained the best results. Since this scene is not perfectly aligned with the X-Z axis (it is slightly rotated around the Y axis), Ahmed et al.'s performed badly on this dataset, only being able to detect one cylinder along the Y axis. Both RANSAC based techniques - Schnabel et al.'s and Liu et al.'s - detected spurious cylinders.

For the *Petrochemical plant* dataset, our technique was able to detect 5 out of the 8 cylinders. An occlusion split one cylinder into two sufficiently far apart from each other, and each part had an arc lesser than 90° during the circle detection procedure. Nevertheless, our technique still obtained the highest F1-score. Once again, RANSAC-based techniques detected planar surfaces as spurious cylinders. Ahmed et al's technique achieved the highest precision, but also detected a large spurious cylinder. Tran et al's detected 3 out of the 8 cylinders, and had the lowest recall.

The *Boiler room* is the most complex dataset and proved to be the hardest among all five. For such dataset, our technique achieved the best precision, recall, and F1-score. It was able to detect 12 out of 21 cylinders. For comparison, Ahmed's, Schnabel, Liu's, and Tran's techniques detected 0, 1, 1, and 2 cylinders, respectively.

For all five datasets, our technique obtained the best F-1 score, demonstrating its

Figure 5.12 – Cylinders 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 cylinders have been highlighted using different colors. Black dots represent samples treated as outliers by each technique.

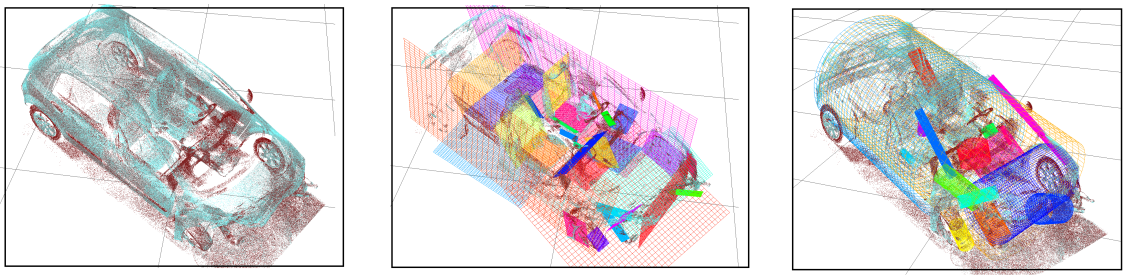


Source: the author.

superior accuracy. We should emphasize that for the experiments reported in Table 5.2, we have fine tuned the parameters of each competing technique for the individual datasets in order for them to obtain their best results in each case. For our technique, on the other hand, we used the same set of default parameter values for all datasets, demonstrating its robustness and independence of parameter tuning. Also, in reverse engineering, recall is more important than precision, since removing spurious structures requires less effort than recreating non-detected structures. We were able to detect much more cylinders in the Boiler Room dataset than any other dataset, and for other dataset we were on par with existing techniques in relation to the number of cylinders detected.

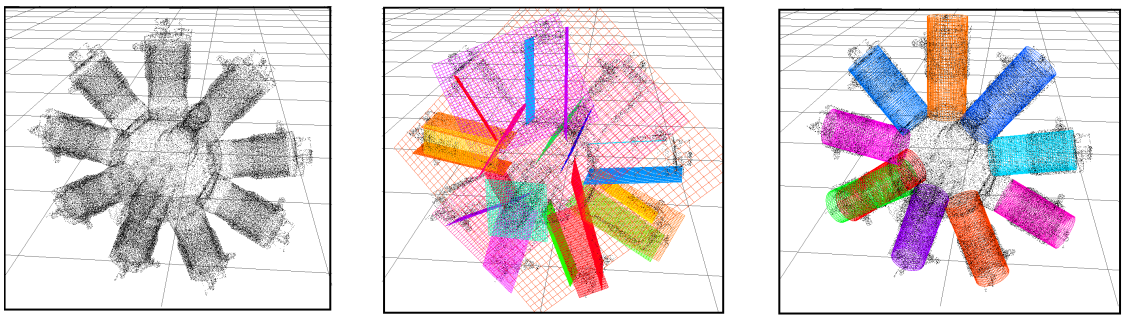
Although our technique is not the fastest (Schnabel et al.'s being first), it is much faster than Tran et al.'s and Ahmed et al.'s. Another point to be stressed is that our technique is deterministic, unliked RANSAC-based solutions such as Schnabel et al.'s and Liu et al.'s, whose results may very among multiple executions on the same dataset.

Figure 5.13 – Detection of planes (b) and cylinders (c) in a point cloud representing a car.



Source: the author.

Figure 5.14 – Detection of planes (b) and cylinders (c) in a point cloud representing a jet engine.



Source: the author.

5.13 Interation with plane detection

We performed two experiments to analyze how our cylinder detection method interacts with the plane detection technique proposed in Chapter 4. For the first experiment we used a real scan from a car, as shown in Figure 5.13 (a), while for the second one we used a synthetic model of a jet engine, as shown in Figure 5.14 (a). Figure 5.13 (b) and (c) display the results of the plane and cylinder detection, respectively, for the car dataset. Interestingly enough, for the cylinder detection, the whole car was considered a cylinder, since it has rounded edges. The plane detection was satisfactory, preserving the planes of the scene, with special attention to the seats. In this particular case, it would be better to first detect planes and then cylinders, since the car is better represented as a set of planar regions than as a single cylinder.

For the jet engine point cloud, however, the cylinder detection was much better

than the plane detection, because this scene is mostly composed by cylinders, as shown in Figures 5.14 (b) and (c). Unlike the car scene, for this scene it would be best to first detect cylinders and then planes, since the plane detection technique is prone to detecting planes on the cylinder surfaces. This was also what we observed in most point clouds extracted from industrial sites, as it is common for those scenes to contain lots of tubular structures. Therefore, in our reverse engineering pipeline, we advise first detecting cylinders and then planes, ocasionaly swapping the order on a case-by-case basis.

6 GRAPHICAL USER INTERFACE

A reverse-engineering framework should accommodate some tolerance to errors generated during the detection process. In order to enable such manual refinement, we developed a graphical user interface to allow one to remove, merge, and semi-automatically detect planes and cylinders. We also present a strategy to detect simple connections between cylinders.

6.1 Plane Refinement

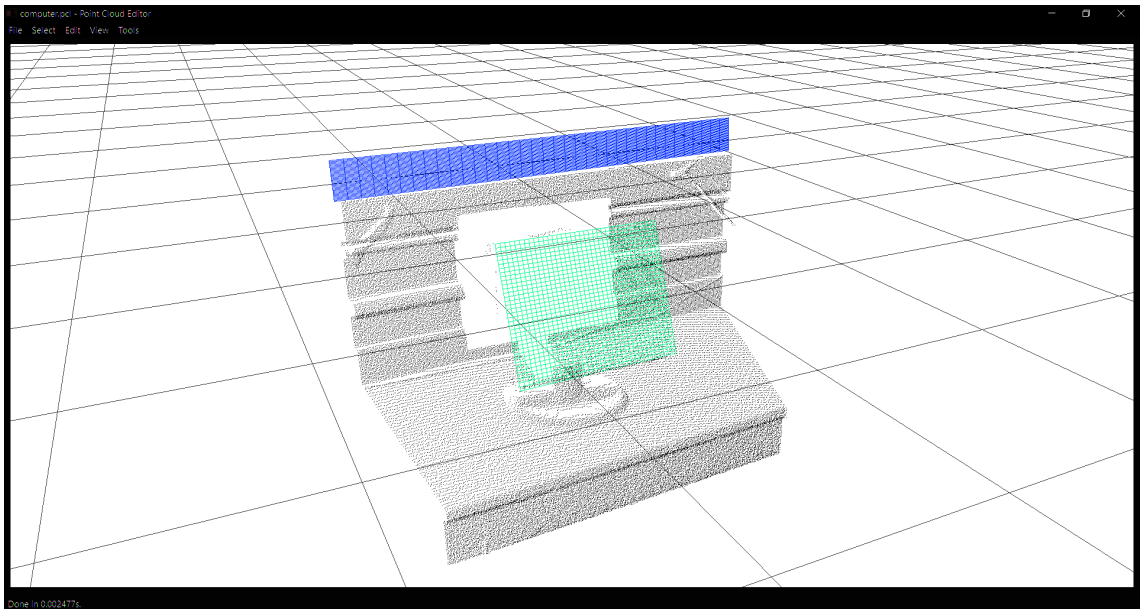
Although our automatic plane-detection technique presents good results in practice, it is not perfect. For a reverse-engineering system to work well, it must allow the user to intervene in the process, deleting spurious planes detected by the automatic technique, and guiding the inclusion of missing ones. Figure 6.1 displays the graphical interface, which allows the user to rotate, translate, or scale the scene using the mouse or keyboard keys. It shows a representation of the point cloud and the detected planes.

For plane detection, the interface provides three options: (i) *delete plane* (ii) *expand selected region* and (iii) *detect plane from selected region*. First, the user selects a region of interest using the mouse. Based on the initial and final positions of the mouse, the software determines the selected samples (the ones whose projections are within the screen rectangular boundaries defined by the initial and final mouse positions). Then any of the three operations mentioned above can be performed on the selected samples.

If the user performs a *delete plane* operation, we check if any select sample belongs to a detected plane. If so, the plane is deleted from the scene. In case the user performs a *detect plane from selected region* operation, the software estimates a plane from the selected points using the the procedure described in 4.3, taking the median of the sample positions and normals.

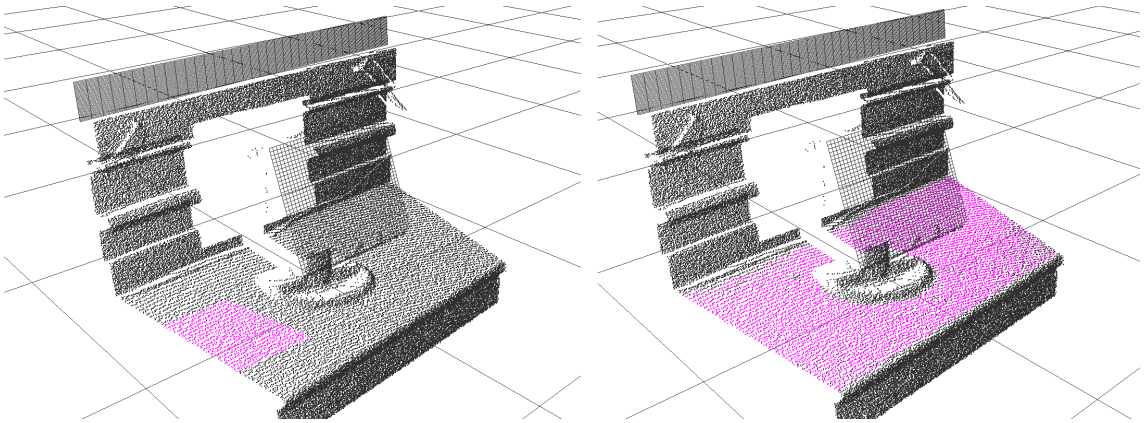
The purpose of the *expand selected region* command is to make the *detect plane from selected region* operation easier, so the user will not need to select the exact region which delimiting the plane, as shown in Figure 6.2. In order to expand the region, the selected region is treated as a patch, and the procedure described in 4.4 is applied to grow it.

Figure 6.1 – Graphical user interface to allow refinement and clean-up of detected planes and cylinders.



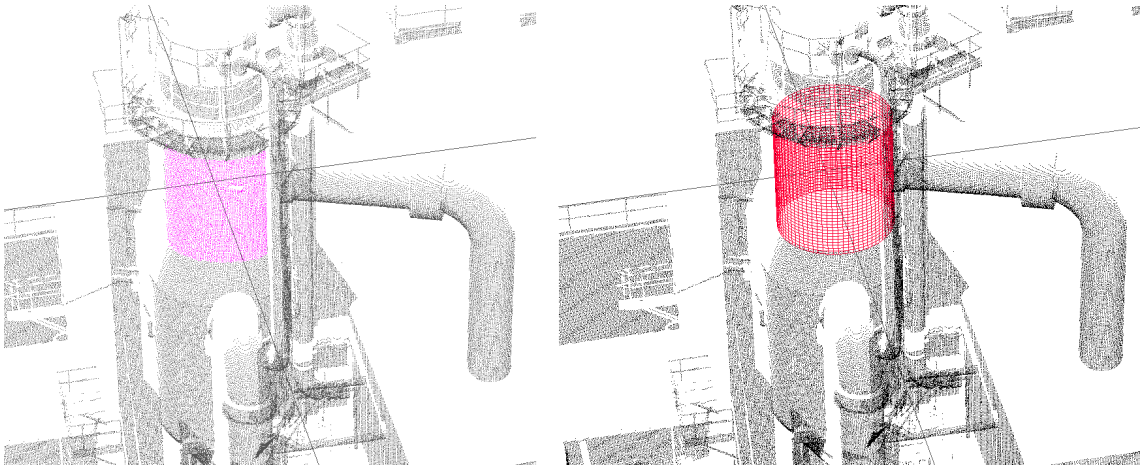
Source: the author.

Figure 6.2 – Example of our *Expand Selected Region* operation. Left: original selected region, marked in purple. Right: region after expansion. This operation allows users to semi-automatically detect planes with precision.



Source: the author.

Figure 6.3 – Example of our cylinder refinement. Left: selected region marked in purple. Right: the detected cylinder.



Source: the author.

6.2 Cylinder Refinement

The same graphical interface used for plane refinement described in Section 6.1 allows cylinder-detection refinement. Similar to plane refinement, the interface provides three options: (i) *delete cylinder* (ii) *merge cylinders* and (iii) *detect cylinder from selected region*.

The *delete cylinder* operation is very similar to the *delete plane* functionality described in 6.1. From the selected samples, the software checks if any sample belongs to any cylinder. If so, it deletes the cylinder. The *merge cylinder* operation removes the selected cylinders and uses the least-squares algorithm presented in Section 5.7 to detect a new cylinder considering all samples belonging to all selected cylinders. The same strategy is applied to the *detect cylinder from selected region* operation, where least squares is used to detect a cylinder from the selected region. An illustration of this process is shown in Figure 6.3.

7 CONCLUSION

This thesis presented the first steps in implementing a general reverse-engineering framework for industrial site plants. We presented two fast and robust techniques to detect planes and cylinders. We also presented a graphical interface that allows the user to further refine the detected structures.

In our automatic plane detection, we presented a robust $O(n\,log\,n)$ technique 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 are 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 a great level of precision and detail. Our solution is robust to noise and virtually independent of parameter tuning.

We demonstrated the effectiveness of our plane detection 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 plane detection technique, on the other hand, automatically adjusted its parameter values based on the local sample distribution.

We also presented a fast and robust technique for automatic detection of cylinders with arbitrary orientations in unorganized point clouds. It consists of orthographically projecting the point cloud along with multiple directions and refining them, detecting circular projections, removing outliers, fitting cylinders to connected components in 3D, and merging them when appropriate.

We demonstrated the effectiveness of our cylinder detection approach by performing a detailed comparison with the most popular as well as with the most recent approaches for cylinder detection. Such techniques were evaluated on five datasets chosen to stress different aspects and configurations found in real environments. Our technique achieved the best F1-score on all datasets. For these experiments, the parameters used

by the competing techniques were individually tuned for each dataset in order to produce their best results in each case. For our technique, on the other hand, we used the same set of default parameter values for all datasets, showing its robustness and independence to parameter tunning, and ability to handle point clouds in general.

7.1 Limitations

For the automatic plane detection, 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). To avoid this problem, *we perform the cylinder detection before the plane detection*.

Our automatic cylinder-detection technique also has limitations. Currently, during the analysis of the connected components, we only check if the normal of each sample is perpendicular to the normal of the projection plane. This can add undesired samples to the connected component, as shown in the Oil Refinery dataset, where the cylinders were overextended using some near-by samples from the plane (whose normals are also perpendicular to the axes of the two cylinders). A further verification would need to be done in order to disassociate such samples from the cylinder.

One disadvantage of our automatic cylinder detection technique with respect to RANSAC- and Hough-transform-based solutions is that, if a cylinder is fragmented into disconnected sections whose projections form arcs smaller than 90° each, the cylinder will not be detected, as each arc does pass the circle recognition test. We plan to address the issue shortly.

One point which needs to be stressed is the neighborhood size k used during the building of the neighborhood graph, described in the Section 2.1.2, which is used by both plane and cylinder detection techniques. In all of our experiments, we set $k = 50$, but there may be cases where this parameter needs to be adjusted. If it is set to low, regions that are 'visually' connected will be disconnected in the graph, which may impact the plane (fragmenting it into multiple planes) and the cylinder detection (not being able to detect a cylinder because each fragment could have less than a 90° in the projection). If it is set to high, it will have a negative impact in the performance of both algorithms.

7.2 Future work

Although we presented important initial steps towards the reverse engineering of industrial sites, much work still needs to be done. For instance, it is necessary to detect other primitives common to industrial sites, such as spheres and cones. Also, it is crucial to detect the connection between pipes, such as valves, T-junctions, elbows, unions, etc., since the pipe-run information of an industrial site is one of the core components of a CAD project. Ideally, such structures could be detected using a machine-learning techniques, such as deep learning.

Our automatic plane detection technique can detect false planes in cylindrical surfaces with large radius. Currently, we avoid this issue by detecting cylinders before detecting planes. However, 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. Regarding our automatic cylinder detection technique, a better connectivity check algorithm would need to be implemented to discard cases where adjacent planes to the cylinder surface are considered part of the same component, and cases where cylinders cannot be detected due to lack of local connectivity. This second issue could be solved if we worked with the point cloud in multiple resolutions through the use of an octree, where fine details would be lost in some resolutions but the connectivity requirement would cylinder detection techniques imposes would be satisfied.

REFERENCES

AHMED, M. F.; HAAS, C. T.; HAAS, R. Automatic detection of cylindrical objects in built facilities. **Journal of Computing in Civil Engineering**, American Society of Civil Engineers, v. 28, n. 3, p. 04014009, 2014.

ALEXANDRESCU, A. Fast deterministic selection. In: **16th International Symposium on Experimental Algorithms**. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany, 2017. p. 24:1—-24:18. Available from Internet: <http://drops.dagstuhl.de/opus/volltexte/2017/7612/>.

BALLARD, D. H. Generalizing the hough transform to detect arbitrary shapes. **Pattern recognition**, Elsevier, v. 13, n. 2, p. 111–122, 1981.

BESL, P. J.; MCKAY, N. D. Method for registration of 3-d shapes. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. **Sensor Fusion IV: Control Paradigms and Data Structures**. [S.l.], 1992. v. 1611, p. 586–607.

BOLLES, R. C.; FISCHLER, M. A. A ransac-based approach to model fitting and its application to finding cylinders in range data. In: **IJCAI**. [S.l.: s.n.], 1981. v. 1981, p. 637–643.

BORRMANN, D. et al. The 3d hough transform for plane detection in point clouds: A review and a new accumulator design. **3D Research**, Springer, v. 2, n. 2, p. 3, 2011.

CHAPERON, T.; GOULETTE, F. Extracting cylinders in full 3d data using a random sampling method and the gaussian image. In: **Vision Modeling and Visualization Conference 2001 (VMV-01)**. [S.l.: s.n.], 2001.

EIGEN C++ Library for Linear Algebra. <http://eigen.tuxfamily.org/index.php?title=Main_Page>. Accessed: 2018-07-01.

FARID, R. Region-growing planar segmentation for robot action planning. In: SPRINGER. **Australasian Joint Conference on Artificial Intelligence**. [S.l.], 2015. p. 179–191.

FERNANDES, L. A. F.; OLIVEIRA, M. M. Real-time line detection through an improved hough transform voting scheme. **Pattern recognition**, Elsevier, v. 41, n. 1, p. 299–314, 2008.

FISCHLER, M. A.; BOLLES, R. C. A paradigm for model fitting with applications to image analysis and automated cartography (reprinted in readings in computer vision, ed. ma fischler,". **Comm. ACM**, v. 24, n. 6, p. 381–395, 1981.

HOPPE, H. et al. **Surface reconstruction from unorganized points**. [S.l.]: ACM, 1992.

HOUGH, P. V. **Method and means for recognizing complex patterns**. [S.l.]: Google Patents, 1962. US Patent 3,069,654.

HUBER, P. J.; RONCHETTI, E. M. **Robust Statistics (Wiley Series in Probability and Statistics Book 693)**. Wiley, 2011. Available from Internet: <https://www.amazon.com/Robust-Statistics-Wiley-Probability-Book-ebook/dp/B005PS4HGU?SubscriptionId=

AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=B005PS4HGU>.

KEINERT, B. et al. Spherical fibonacci mapping. **ACM Transactions on Graphics (TOG)**, ACM, v. 34, n. 6, p. 193, 2015.

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.

KIRYATI, N.; ELDAR, Y.; BRUCKSTEIN, A. M. A probabilistic hough transform. **Pattern recognition**, Elsevier, v. 24, n. 4, p. 303–316, 1991.

LEICA Cyclone/Cloudworx Example Databases. <https://hds.leica-geosystems.com/en/29453.htm>. Accessed: 2018-07-01.

LI, L. et al. An improved ransac for 3d point cloud plane segmentation based on normal distribution transformation cells. **Remote Sensing**, Multidisciplinary Digital Publishing Institute, v. 9, n. 5, p. 433, 2017.

LIMBERGER, F. A.; OLIVEIRA, M. M. Real-time detection of planar regions in unorganized point clouds. **Pattern Recognition**, Elsevier, v. 48, n. 6, p. 2043–2053, 2015.

LIU, Y.-J. et al. Cylinder detection in large-scale point cloud of pipeline plant. **IEEE transactions on visualization and computer graphics**, IEEE, v. 19, n. 10, p. 1700–1707, 2013.

MITTAL, S.; ANAND, S.; MEER, P. Generalized projection-based m-estimator. **IEEE transactions on Pattern analysis and machine intelligence**, v. 34, n. 12, p. 2351, 2012.

MOORE, A. W. An intoductory tutorial on kd-trees. Citeseer, 1991.

PHAM, T. T. et al. Geometrically consistent plane extraction for dense indoor 3d maps segmentation. In: IEEE. **Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on**. [S.l.], 2016. p. 4199–4204.

RABBANI, T.; HEUVEL, F. V. D. Efficient hough transform for automatic detection of cylinders in point clouds. **Isprs Wg Iii/3, Iii/4**, v. 3, p. 60–65, 2005.

ROUSSEEUW, P. J.; CROUX, C. Alternatives to the median absolute deviation. **Journal of the American Statistical association**, Taylor & Francis, v. 88, n. 424, p. 1273–1283, 1993.

ROUSSEEUW, P. J.; DRIESSEN, K. V. A fast algorithm for the minimum covariance determinant estimator. **Technometrics**, Taylor & Francis Group, v. 41, n. 3, p. 212–223, 1999.

SCHNABEL, R.; WAHL, R.; KLEIN, R. Efficient ransac for point-cloud shape detection. In: WILEY ONLINE LIBRARY. **Computer graphics forum**. [S.l.], 2007. v. 26, n. 2, p. 214–226.

TRAN, T.-T.; CAO, V.-T.; LAURENDEAU, D. Extraction of cylinders and estimation of their parameters from point clouds. **Computers & Graphics**, Elsevier, v. 46, p. 345–357, 2015.

VO, A.-V. et al. Octree-based region growing for point cloud segmentation. **ISPRS Journal of Photogrammetry and Remote Sensing**, Elsevier, v. 104, p. 88–100, 2015.

WENG, J.; ZHANG, Y.; HWANG, W.-S. Candid covariance-free incremental principal component analysis. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 25, n. 8, p. 1034–1040, 2003.

WRIGHT, J. et al. Robust principal component analysis: Exact recovery of corrupted low-rank matrices via convex optimization. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2009. p. 2080–2088.

XU, L.; OJA, E.; KULTANEN, P. A new curve detection method: randomized hough transform (rht). **Pattern recognition letters**, Elsevier, v. 11, n. 5, p. 331–338, 1990.

ZOU, H.; HASTIE, T.; TIBSHIRANI, R. Sparse principal component analysis. **Journal of computational and graphical statistics**, Taylor & Francis, v. 15, n. 2, p. 265–286, 2006.