Ana Maria Stănescu

Semantic Segmentation of Dense 3D Point Clouds with Geometric Primitives

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieurin

Master's degree programme
Computer Science

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Dieter Schmalstieg
Institute of Computer Graphics and Vision

Advisor

Dipl.-Ing. Dr.techn. Clemens Arth
Institute of Computer Graphics and Vision

Graz, Austria, April 2018

Computer science inverts the normal. In normal science you're given a world and your job is to find out the rules. In computer science, you give the computer the rules, and it creates the world.

Alan Kay

Abstract

This thesis presents an approach for structural modeling from dense unstructured 3D point clouds. The core contribution is an efficient method for fitting geometric primitives based on Support Vector Machines. Dense 3D point clouds are acquired together with color images on a mobile device with an attached depth sensor. Then, after a surface normal-based segmentation, geometric primitives that describe the geometry of the point cloud are robustly estimated: planes, spheres and cylinders. The fit is refined using non-linear optimization. Close similar primitives are merged together, simplifying the representation of the scene. This is followed by an evaluation based on classification of features encoding the quality of the fit. The approach iterates over successive frames to optimize the fitting parameters or replace a detected primitive by a better fitting one. As a result, we obtain a semantic model of the scene consisting of a set of geometric primitives. We evaluate the approach on an extensive set of scenarios and discuss the results, measuring the primitive detection performance in terms of precision and recall. Finally, we present thoughts on the future improvements to the method.

Kurzfassung

Diese Arbeit stellt eine Methode für das strukturelle Modellieren von dichten unstrukturieren 3D Punktwolken vor. Der Kern dieses Verfahrens beschäftigt sich mit dem effizienten Schätzen von geometrischen Primitiven welches mit Hilfe von Support Vector Machines erfolgt. 3D Punktwolken mit dazugehörigen Bildern werden mit einem Mobilgerät aufgenommen, das mit einem Tiefensensor bestückt ist. Nach einem normalvektorbasierten Segmentierungsverfahren werden Ebenen, Sphären und Zylinder in der Szene robust geschätzt. Der Fit der Objekte wird mit einer nicht-linearen Optimierung verbessert, und gleichzeitig werden nahestehende ähnliche Objekte miteinander verschmolzen, was die Repräsentation der Szene vereinfacht. Daraufhin wird eine Evaluierung verschiedener Eigenschaften vorgenommen, welche das Qualitätsmaß des Fits für die einzelnen Objekte wiederspiegeln. Diese Methode verarbeitet eine Reihe von aufeinanderfolgenden Bildern und optimiert gleichzeitig die Parameter der Schätzung der Objekte. Dies kann dazu führen, dass bereits detektierte Objekte durch besser approximierte Objekte ersetzt werden. Daraus ergibt sich ein sematisches Model, welches die Szene beschreibt und aus geometrischen Primitiven besteht. Wir evaluieren die beschriebene Methodik anhand von zahlreichen Beispielszenen und erörtern die Ergebnisse anhand der Trefferquote und Genauigkeit. Zuletzt werden Verbesserungsvorschläge für diesen Ansatz erläutert.

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

The text document uploaded to TUGRAZonline is identical to the present master's thesis dissertation.

Date	Signature

Acknowledgments

Firstly, I would like to thank Dipl.-Ing. Dr.techn. Clemens Arth for the advice and the guidance. Your enthusiasm and ideas amplified my motivation and my curiosity.

Thank you Univ.-Prof. Dipl.-Ing. Dr.techn. Dieter Schmalstieg for your support. It is an inspiration to work in an environment of passionate people such as your group.

I am very grateful to Rafael Roberto for his dedication and patience when introducing me to this topic. Your work and our discussions unraveled for me new perspectives in the field that I have decided that I will further pursue.

I want to express my gratitude towards Christoph Klug for the thorough explanations from which I have learned a lot. Our conversations were always very helpful and interesting.

Also, I would like to thank Philipp Fleck for the ideas and the help, and my office colleagues Mina Basirat and Evelyn Gutschier for the encouragement. I am grateful to all those who were by my side in the awesome experience that brought me to the end of my studies.

My greatest thanks go to my family, especially to my mother and to my grandfather Mihai for always believing in me.

Thank you Filip for your tireless support.

Contents

List of Figures

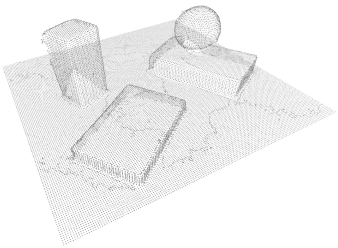
List of Tables

1 Introduction

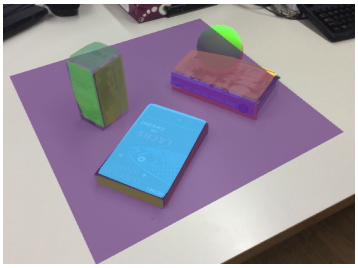
Contents

1.1 Motivation

(a) Input RGB image from the camera.



(b) Dense unstructured 3D point cloud as captured by the Structure sensor.



(c) Overlay of detected primitives in a video for a live-view AR application.

Figure 1.1: A captured point cloud with the corresponding color image and the primitives detected by the proposed method superimposed on the color image.

The availability of dense 3D point clouds with registered RGB images allows for entirely new levels of understanding of the environment and opens new ways to interact with it. The ultimate goal would be that the objects from which the 3D points are sampled from are fully recovered with their spatial and semantic properties. It can be regarded as two separate tasks: segmentation and classification. Firstly, objects or regions of interests need to be delimited. Secondly, they need to be identified as concrete object categories.

Modeling real environments with polygonal meshes and other geometric primitives is often referred to as *structural modeling*. 3D depth sensors acquire the approximate shape of the underlying scene in the form of a dense 3D point cloud. These point clouds have to be further processed to generate meshed surfaces. Despite a high computational effort, a meshed surface usually has no semantic meaning. However, our everyday environment largely consists of very few geometric primitives, mostly planes, boxes, cylinders and spheres. This prior knowledge can support mesh generation in a significant way. If a point cloud can be transformed into a set of geometrically meaningful entities, Augmented Reality (AR) interactions, like placing virtual objects or automatic object highlighting in the environment, become feasible.

Fitting geometric primitives for structural modeling has been investigated in various contexts. Examples include robot grasping and robot localization [1], collision detection [2–4], architectural modeling [5–8], 3D reconstruction [9], and scene denoising and compression [9]. Extensions to AR [10–12] have a stronger emphasis on real-time aspects and online acquisition of input data. Most methods build on RANSAC [13] or Hough transform [14] for robust primitive fitting.

Semantic modeling from real-world assemblies is supported in some commercial modeling products, but usually requires substantial manual interaction. For example, Curvsurf[1] requires manual labeling of 3D point clouds. Based on the fit of a primitive, parts of the geometric primitive are meshed, and points replaced. Although this approach gives reasonable results, choosing geometric primitives, scaling them and finally aligning them in 3D is tedious. Clearly, the ability to automatically determine geometric primitives in dense 3D point clouds has high potential for scientific and commercial use.

In the proposed method, we aim to segment a dense, unstructured, noisy point cloud captured by a 3D sensor over time, by fitting 3D primitives to it: planes, spheres and cylinders. The choice of representing the environment with primitives is based on the idea that the majority of man-made objects can be modeled with these shapes [3, 15]. We can also argue that the primitives carry semantic meaning, in the sense that, for example, in a tabletop scenario, planes can be a book or the table surface, while cylinders can be bottles, pens or cups. In an industrial scenario, identifying cylinders could be equivalent to finding pipes and levers.

In comparison to approaches that segment or classify concrete object instances, in the case of fitting geometric primitives, the model carries less specific semantics. This is why the primitives can approximately describe anything, independent of size or pose, and specific, restricting assumptions about the scene are not needed. The result is a lightweight generalized representation of the scene, as shown in Figure 1.1. Here an image of a scene is exhibited together with the corresponding point cloud captured by a depth sensor. Figure 1.1c shows the results of the proposed method.

[1] `http://www.curvsurf.com/`

1.2 Outline

First, approaches related to structural modeling and semantic segmentation of a 3D scene are presented in Chapter 2. We identify three main directions in this research area: geometric methods, Machine Learning (ML) methods, and a mixture of the two. This is accompanied by a short discussion regarding advantages and disadvantages of the methods. We also argue the relevance of our approach in its field, compared to similar approaches.

Chapter 3 outlines the proposed algorithm and offers an overview of the pipeline. Furthermore, the role of the individual steps is specified with an emphasis on their interdependence.

The stages of the proposed structural modeling method are explained in Chapter 4. We provide details about the choice of algorithms and discuss their parametrization.

A classification approach of primitives based on their parameters as well as the definition of the used features are presented in Chapter 5. We offer an insight on labeling training data and on the properties of the data. Furthermore, experimental conclusions regarding the used classifiers are presented.

The obtained results are discussed in Chapter 6. The acquisition setup and properties of used datasets and equipment are described, visually exemplifying captured point clouds. In order to test our approach on noiseless data, we manually create artificial datasets which are presented in this chapter. We show an evaluation of the proposed algorithm on both real and artificial datasets in terms of precision and recall regarding the correct identification of primitives. A short description of the used software and hardware tools is also provided.

In Chapter 7, the limitations of our approach are exemplified and discussed. Solutions, prospective improvements and future work are brought to attention. Finally, we draw conclusions regarding our work.

2 Related Work

Contents

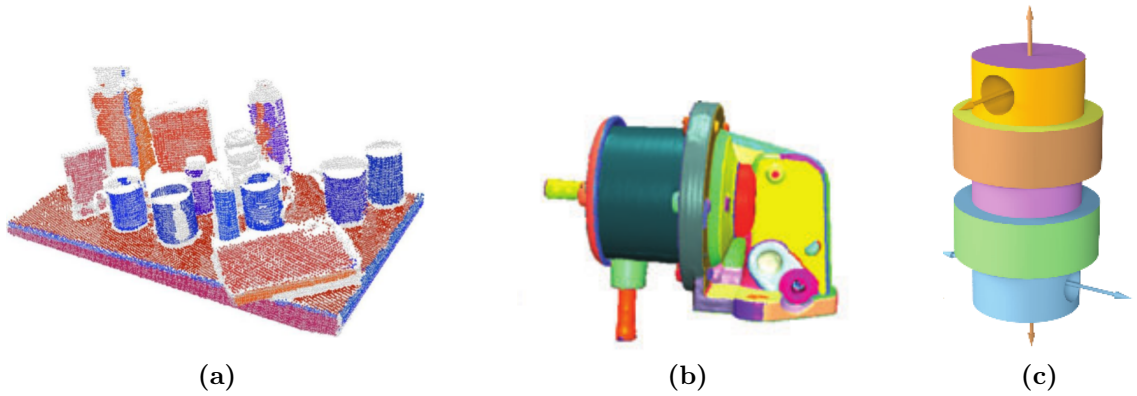
The borders between segmentation and classification are ambiguous, as it is a chicken-and-egg problem: one has to classify objects in a scene, but, in order to partition the scene into regions with similar features, one has to know the partitioning criterion. Thus, segmentation and classification are tightly connected. We present methods relevant for the context of the presented work. We focus particularly on geometric segmentation by using geometric primitives, as it is the direction that our method follows.

2.1 Geometric Approaches

A method dealing with efficient segmentation of unorganized point clouds with RANdom SAmpling Consensus (RANSAC) [13] is introduced by Schnabel et al. [9]. It is based on iteratively fitting five geometric primitives, plane, sphere, cylinder, cone and torus, to a point cloud in an efficient way, without global relations between the primitives. They propose an effective sampling strategy for RANSAC that maximizes the likelihood of convergence in a small amount of steps. Each of the five primitive categories is fit to the point cloud and the quality of the fit is described by a score, according to multiple criteria like number of inliers, curvature, connectivity. The points belonging to a fit shape are removed, and the algorithm continues performing fitting. It is also suited for noisy data, as their results show.

Rusu et al. [2] propose an approach to reconstruct and segment a scene by fitting geometric primitives to a point cloud: planes, spheres, cylinders and cones. In order to

capture object details, small regions that do not fit into the parametric models are modeled with meshes, obtaining a hybrid model. The assumption of the existence of a horizontal planar table is considered. First, the scene is split into table inliers, by fitting planes and clusters supported by the table. An octree connectivity segmentation is performed to segment and fit primitives to clusters. After checking intersections, the model with the most inliers is chosen. Each model is refined with non-linear optimization. Afterwards object handles are identified by triangulating surface areas defined by points neighboring the primitives. The outliers that are left after the model fitting step are also triangulated and added to the shape.



(a) (b) (c)

Figure 2.1: Examples of point cloud segmentation with primitive fitting from related literature: [2] - left, [9] - middle, [16] - right.

Holz et al. [3] introduce a method to segment planes from RGB-D images in real-time, for a mobile robot, in the context of collision avoidance and object grasping. This approach focuses on performance, and has two main stages: calculating surface normals from depth images and clustering the normals into plane segments. The clustering algorithm is based on comparison of normal orientation in-between space voxels. When extracting semantics needed for the functionality of the robot, this method assumes that the vertical direction of the scene is known, so that the floor can be identified. RANSAC is used to refine plane fitting robustly. Object candidates are detected by projecting points on the convex hull of plane candidate inliers, by assuming that their size is between 1 and 10 centimeters, so that the robot can grasp them. The approach runs in real time and manages to detect obstacles with 100% accuracy, as long as they are neither too small nor far away from the sensor.

A method using RANSAC for primitive fitting that focuses on global relations between the fit primitives is proposed by Li et al. [16]. First there is an initial RANSAC primitive fitting, where each found primitive together with its inliers is removed from the point cloud before the next primitive is fit. Then a graph is built that contains relations between the primitives such as perpendicularity and parallelism. The relations are optimized and more complex behaviors are modeled such as angles between primitives; also equality checks

are performed. Finally RANSAC and primitive fitting are run again on the remaining points, and a re-alignment step is performed. The drawback of the method is that it is computationally expensive.

Another method that focuses on the 3D segmentation of a tabletop scenario is introduced by Ückermann et al. [4]. The algorithm uses depth maps and has two stages: splitting the scene into regions delimited by strong variations in orientations of surface normals and further dividing of the clusters into object hypotheses. Using depth maps, angles between neighboring surface normals are calculated for each image pixel, in eight directions. The smallest angle between normals is taken into account to decide if there is an object edge region. The point cloud regions in between the edges are then clustered together by means of a region growing algorithm. There is the assumption that an indoor scene contains large planar surfaces: tabletops, walls, floor. Therefore, RANSAC is used to fit planes to clusters with large enough number of points. Unprocessed points are associated with their closest plane segment, and very small clusters are discarded. Then, clusters are merged together if they are close to each other. The algorithm displays stability, decreasing in performance and increasing in run-time when dealing with cluttered regions.

An approach related to the one by Ückermann et al. [4] was introduced by Tateno et al. [17], focusing on a semantic segmentation of an indoor scene in real time with Simultaneous Localization and Mapping (SLAM). The method covers two aspects, SLAM reconstruction and the scene segmentation into a Global Segmentation Map (GSM). The latter is based on the idea that objects in the real world are convex and delimited by concave regions. Mathematical quantities are introduced to measure such regions for separating convex objects, similarly to criteria in [4] for finding 3D object edges. Afterwards points composing object surfaces are clustered together by means of a connected component algorithm. The object labels are propagated from frame to frame by projecting segmented objects onto the image plane and comparing their labels with the labels of the current segmentation. Objects that most likely belong to the same label are merged based on a confidence scheme. This approach is effective, as only visible objects are segmented and used for updates each frame, so the complexity remains constant. It does not use primitive fitting; the results are obtained only by splitting the scene into convex objects.

Hettiarachchi et al. [12] use 3D primitive fitting in an AR context. Virtual objects are matched to real objects approximated by 3D primitives to enhance a user's experience when handling them, by providing haptic feedback. After the horizontal support plane is identified in the scene, point clusters are segmented with a clustering algorithm. Primitives are fit to the point clusters using the approach proposed by Schnabel et al. [9]. The detected primitives are compared to user-defined objects composed of primitives, taking into account similarities between the parameters of the primitives. The similarities per object are quantified into a score. Then a global maximum for the whole scene is found in terms of this score. The identified primitives with overlaid virtual objects are tracked in real-time, in an AR application. Results about the detection rate of the primitives are not

mentioned, as the paper focuses on user experience, presenting the successful outcome of a user study in terms of usability.

Our method is the most similar to the ones proposed by Roberto et al. [18], where a primitive fitting of planes, spheres and cylinders is proposed. After fitting primitives in a sparse noisy point cloud with the Efficient RANSAC of Schnabel et al. [9] in a certain keyframe, the best fit for a shape is chosen by looking at the mean distance from the point cloud to the projected points on the primitive. Similar primitives are merged. Afterwards, a check is performed to decide if a shape is good enough, based on criteria such as number of inliers, dispersion of points on the surface, average distance to the primitive and accepted radius, in the case of spheres and cylinders. The method keeps track of detection history, such that primitives that are detected more often and more precisely are kept, while unstable primitives are discarded. It achieves a better F1-score than Schnabel et al. [9] in all tested scenarios.

Roberto et al. [19] propose another version of their algorithm, with improvements such as a computation of a detection history-based score for deciding the best fitting primitive. This score is also used for shape recovery. This involves validating a shape that was rejected in previous frames. For the tested scenarios, 100% precision is achieved, showing an improvement in comparison to the previous version of the algorithm.

Oesau et al. [6] introduce a method for segmenting a large-scale indoor point cloud representing building floors into categories such as floor, ceiling and walls, using RANSAC and the Hough transform [14]. The point cloud is projected along the vertical world direction, considering that regions with dense projection indicate floor splitting. The dense regions in the 1D projection are identified as ceilings and floors with mean shift in the histogram of 1D point positions. The regions between floor and ceiling are then split into wall slices, where walls are detected, in three stages: the walls in a wall bounding box are projected into the horizontal plane, omitting the points where the normals are not parallel to the horizontal plane, as it is assumed that walls are all vertical. After being downsampled, the 2D projected wall points are used for 2D line fitting with RANSAC. Then they are clustered according to the fit lines and additionally with the bilateral filter on normal directions, and then all 2D lines are detected with the Hough transform. The rooms are obtained by stacking the 2D slices after merging similar segments. Finally, the 3D model of the floors split in rooms is created by labeling the cells between plane intersections as empty or full space as a global energy minimization problem which is solved with a graph cut algorithm. The method can deal with some noise as well, and is able to approximate non-planar surfaces. However, it is limited to vertical walls.

Xiao et al. [7] fit geometric primitives in the form of cuboids to retrieve the plans of museums. They use data with little noise, acquired by repeated closeup laser scans in museum rooms. They estimate the surrounding surfaces by using Inverse Constructive Solid Geometry models. This is a combination of cuboids aligned with the horizontal plane, having any rotation around the vertical world axis. First, horizontal slices are extracted, containing line segments found with the Hough transform. The segments are put together

into candidate rectangles. These are then combined together in a greedy fashion into 2D models. Similarly, a set of 3D candidate cuboids is built, by trying different combinations of stacking 2D rectangles; they are chosen to be part of the final model by making use of a score that measures whether they are part of a surface. Extra post-processing steps of the cuboids are performed to retrieve a realistic floor plan and to apply a texture on the obtained model. This method is specialized on rooms and floor plans. This is the reason for the choice of cuboids to describe the point cloud.

An approach for scene segmentation that handles only ellipsoidal objects in proposed by Georgiev et al. [20]. This implements a horizontal plane-sweep-like method, where 2D ellipses are fit to each horizontal plane slice cutting the scene. They suggest an algorithm that analytically fits multiple 3D spheres and can deal with noise. Afterwards, the 3D objects are reconstructed by checking geometrical properties of the objects resulting from putting the slices together, obtaining spheres, cylinders and cones. The approach runs in real-time and uses a Kalman filter for object tracking and noise reduction, as the Microsoft Kinect introduces systematic noise. However, it is limited to 3D geometric primitives that are based on 2D ellipses.

This kind of methods allow for a generalized abstraction of the scene, less dependent on the type of objects. Nevertheless, some assumptions are made, such as the scene setup, or the vertical orientation of the scene. The choice of primitives is adapted to the context: When trying to retrieve building plans, boxes or orthogonal planes are chosen, while for small, cluttered objects, appropriate categories are planes, spheres, cylinders, cones, torii and others. For industrial scenarios, cylinders are a more appropriate choice. For collision avoidance of a robot, planes are used to estimate the environment.

Depending on the use case, the idea of using primitives can be both an advantage and a drawback. On the one hand, the shapes do not model the scene exactly, depending on the primitive categories chosen and on the type of scene captured. On the other hand, one obtains a very general, compact, straight-forward representation of the scene.

2.2 Machine Learning Approaches

The method introduced by Nan et al. [21] is based on segmenting a scene, classifying scene clusters and inferring semantics by fitting pre-learned 3D deformable object templates. First, the scene is segmented into patches with similar normal orientation. Then, patch triplets composed of nearby patches are expanded by adding neighbor patches. This expanded point set is classified with Random Decision Forests into concrete object categories, then removed from the original point cloud. The features used for the classification are computed by fitting the clusters into a bounding box and segmenting it into three slices along the vertical axis according to the density of the points distribution along this axis. The features are based on the aspect ratio of the three sub-boxes. The classifier is trained with point clouds of common man-made objects. The classification is followed by a template fitting step. After common object templates are overlaid on point clusters,

parts of the template are scaled to fit better in terms of minimizing the Euclidean distance. To preserve the general object shape and properties, a structure-preserving deformation optimization is also run after each deformation step. More templates of the same class are fit. The best one is chosen, and outliers are removed. The method is applied on dense point clouds with little noise and manages to correctly segment and identify the objects most of the time. The approach faces a challenge when the objects are not approximately in vertical position. Another drawback is that the algorithm is not successful when large parts of the objects are missing.

Hackel et al. [22] propose a method for segmenting point clouds of outdoor scenes using kD-trees. The approach focuses on efficiency and achieves it by using multiscale features and efficient data structures for storing and manipulating the point cloud. The features used are purely geometrical, based on properties of the local point neighborhoods at different scales. They are derived from the local 3D structure tensor of a point and its nine nearest neighbors, as well as from a cylindrical neighborhood properties. Other 3D descriptors are tried, such as Signature of Histogram of Orientations [23] and Shape Context 3D [24], but they are proven to be inefficient, while only slightly improving the classification performance. After concluding that points located closer to edge regions carry more information, they are used for the scene segmentation. The segmentation method uses Random Forests as classifiers with five to seven classes depicting concrete object classes: trees, pedestrians, facades and other similar categories. They reach a precision of over 90 percent for classifying Facades, Ground and Car, while the classification of Motorcycle, Pedestrians or Traffic Sign achieves a poorer performance.

As neural networks have become increasingly popular, they have also been used for 3D segmentation. Some recent work include SEGCloud by Tchapmi et al. [25], a 3D point cloud segmentation approach using a joint pipeline composed of a 3D Fully Convolutional Neural Network together with Conditional Random Field (CRF) to predict point-wise class labels. This method is comparable to Machine Learning-based state-of-the-art approaches and partly outperforms them, when measuring classification success on benchmark databases such as Semantic3d.net [26], Stanford Large-Scale 3D Indoor Spaces Dataset [27], NYU v2 [28], or KITTI [29].

Also McCormac et al. [30] use Convolutional Neural Network (CNN) for 3D semantic segmentation, in the context of real-time robotic navigation with a SLAM system. The classification is performed by a CNN on RGBD images, getting a per-pixel probability distribution over the possible classes as an output that is updated between frames incorporating prior predictions in a Bayesian sense. CRFs are used for regularization by making use of the geometric neighborhood between surfels. The results on the NYU v2 [28] dataset show improvements over RGB-only methods using CNN.

Armagan et al. [31] use semantic segmentation of building facades from images for pose estimation and 3D localization. After semantically segmenting images of an outdoor scenes into facades, edges and background by feeding the color images of the scene to a Fully Convolutional Network, they preform geo-localization. For this, the pose is es-

timated at each frame by sampling more candidate poses and checking which yields the highest likelihood in terms of predicted labels at certain positions, from the 2D map of the surroundings.

These approaches use trained models to segment and gain semantic information about a 3D scene. The modeled object is based on prior information about an its inherent properties. This means that the ability of the models to capture an object's distinctive traits and the variety of models is of crucial importance. While this results in more exact, semantically rich model of a scene, it also implies that the category of expected objects in known.

2.3 Mixed Approaches

(a) (b) (c)

Figure 2.2: Examples of point cloud segmentation results from related literature using mixed approaches: [32] - left, [33] - middle, [34] - right.

Rusu et al. [32] propose a CRF-based classification approach for point clouds, using efficient point features based on the geometrical properties of the point neighborhood, namely Fast Point Feature Histogram (FPFH). An indoor tabletop context is assumed. After removing the support plane, a clustering is performed to separate objetcts on the table. Afterwards, FPFH features are calculated for each point. A CRF represents neighborhood relations between points, connected to their computed FPFH features. This model is used together with Support Vector Machine (SVM)s to classify points as belonging to the following primitive-like surfaces: cylinder, edge, corner, torus, plane. The training is run on manually labeled point clouds. Experimental results show better classification accuracy of the CRF over SVM, reaching over 90 percent. This approach runs on low noise laser-scanned point clouds.

Xiong et al. [35] use CRF to model contextual relationships in a laser-scanned point cloud, with the goal of extracting an architectural model of one or more rooms. Regions such as wall, floor, ceiling, and clutter are identified. Similarly to Holz et al. [3], the point cloud is initially approximated by grouping the points into voxels and calculating a normal per voxel, by means of plane fitting in a neighborhood. Then a region growing

algorithm is used to cluster planes with similar orientations, and boundaries are delimited by convex hulls of the resulting planes. Next, a CRF model is built, representing the plane patches as nodes. The nodes use features such as orientation, area and height and pairwise relations such as orthogonality, parallelism, adjacency and coplanarity. The model is trained on dense registered point clouds, which are partially manually annotated. In comparison to other room modeling methods, this approach identifies clutter and does not use the assumption that the rooms are mostly empty. However, the vertical orientation is considered to be known.

Xiong et al. [36] propose an improvement to their method, by identifying doorways and window openings after the classification into wall, floor ceiling and clutter. Instead of CRF, a stacked learning algorithm is used. After clutter removal, raytracing is performed to distinguish between occluded voxels and openings in the walls. The openings are then classified with SVM. A 3D inpainting approach is run to improve the coherence of the labels. The algorithm manages to detect more than 90% of the openings and surpasses the initial classification performance of the previous approach, but still has difficulties with distinguishing clutter from walls. A surface-based model is obtained, which can be further processed into a Building Information Modelling (BIM). This approach uses manually labeled points for training and dense point clouds from professional laser scans with low noise.

Koppula et al. [37] introduce a graphical model based classifier for semantic scene interpretation. First, they over-segment the point cloud based on normal orientation and surface continuity, then classify each obtained segment into concrete object classes. The features used for learning regard individual segments and relations between segments and include not only geometrical and positional properties, but also color-based features. The method is tested on large dense indoor point clouds captured with a Kinect sensor, and shows the relevance of the contextual relationships' impact on the classification performance. The paper also presents experiments with a robot that has to place and find various objects in a scene.

The graphical model approaches make use of the scene context and manage to learn properties of scenes inspired by human perception such as associations between objects that typically occur together, and common positions of objects in a scene. Of course, the trade-off is lack of generality and the fact that they are suitable for specific scenarios.

Huang et al. [34] present a method for point cloud segmentation and classification in an industrial scenario. Here the predominant shapes are planes and pipes. Binary class SVMs are trained to distinguish five classes: plane, pipe, edge, thin pipe; the rest of the points are classified as other. The features used for this procedure are FPFH. After the classification, points are clustered together in point cloud components and removed. The rest is filtered by relevance, and compared to components that already have semantic information associated for further classification, by using other point features, namely 3D self-similarity descriptors [38]. A RANSAC variant is used in this step to calculate a rigid transformation between point clouds. The algorithm is successful at detecting the

discussed categories in cluttered scenes; however, the data used has little noise, and large curved surfaces are mistakenly identified as planes.

Methods have been proposed that try to model a scene by using only planes. Nguyen et al. [33] introduce such a method, that aims to represent a scene by an ensemble of planes with global relations, in the context of a SLAM system. A set of initial planes is found in a region-growing manner, starting with a set of seed locations and fitting planes by minimizing a linear least squares problem in inverse depth representation. Isolated pixels are reassigned to components based on a neighborhood vote. A connected component algorithm is run to re-group the points into individual components. Geometric relations between plane segments are found: parallelism, coplanarity, incidence, orthogonal incidence. Plane segments that overlap are merged into a same-plane cluster, called plane features. The remaining planes also define plane features on their own. Geometric relations between plane features are inferred, then checked. Afterwards, camera poses are optimized together with plane features using the Ceres solver [39]. The scene is stored as a half-edge data structure that describes plane borders. Further refinement steps on the data structure are performed. A final polygon fitting step using Expectation Maximization is performed to further simplify the estimated models. The resulting reconstruction is evaluated by comparison to a ground truth model, showing an accuracy of up to two centimeters. This method runs in real time, but can only approximate planar structures.

Among the Computer Aided Design (CAD) related approaches, Ochmann et al. [8] try to reconstruct an ensemble of rooms from laser-scanned point clouds by using RANSAC based plane fitting, global optimization and extra refinement steps, while keeping track of the rooms in a graph data structure. First, the point cloud is coarsely segmented into rooms, based on the fact that, in general, there is one scan per room. Walls are being assigned a thickness in the form of a parallel plane to the wall plane. Then a plane fitting stage using the RANSAC-based approach introduced by Schnabel et al. [9] is performed. After the wall candidates are obtained, global optimization is used to attribute the best wall labels, and to model the plane adjacencies. The optimized cost function is based on wall properties such as point distribution within a wall surface, or assignment of the points to more than one plane. Points that are not part of the walls are clustered together and classified with SVMs into doors, windows, virtual walls and invalid components, where virtual walls are non-existent walls, created by overlaps. The datasets are captured with professional laser scanner equipment and have negligible noise and high density.

In contrast to the aforementioned method by Oesau et al. [6], Armeni et al. [27] parse large-scale point clouds representing rooms of a building by trying to detect the spaces between walls and floors, by using a projection of the points on a horizontal axis. By analyzing the discrete distribution of the projected points with signal processing techniques, a partitioning in rooms is obtained. Furthermore, objects in the rooms are classified inside of a 3D sliding window using a pipeline of binary SVMs for a set of candidate class labels withing a voxel. CRF is used for maximizing the likelihood of the labels being found next to each other, together with the confidence score from the SVM. The results are compared

to two geometric approaches, [7, 9], exceeding their mean detection precision on labeled benchmark point cloud datasets.

For more details, Chen et al. [40] offer an overview of 3D indoor scene modeling methods, up to 2015, including comparisons and discussions. Another general overview of existing 3D segmentation methods up to 2017 is presented in the work of Grilli et al. [15].

2.4 Context

As discussed, ML techniques such as SVM [8, 32, 34] and probabilistic graphical models [32, 35, 37] have been used for structural modeling of geometric primitives. Our proposed approach uses a segmentation-fitting-refinement pipeline [2, 18, 33], applied over multiple frames to detect primitives. SVM classification is used to discard outdated shapes. In contrast to previous work [2, 32], our method is designed to improve the structural modeling in time over multiple frames. Unlike some related work [18], we focus on dense point clouds. Also, in a particular frame, we employ ML to decide where refitting of a primitive is necessary. Therefore, our approach introduces a novel criterion for reasoning about the quality of the fit, which is analyzed in this thesis.

Table 2.1 shows an overview of the discussed related work, with relevant information such as the models used for scene modeling, the scene context, the type of data used, and a rough estimation of the noise level. The last table entry represents our proposed approach for comparison.

Year	Reference	*Primitives*	*Plane*[1]	*Sphere*	*Cylinder*	*Cone*	*Torus*	*Global rel.*	*RANSAC*	*ML*[2]	Context	Dataset	*Noisy*[3]	*Multiframe*[4]
2007	Schnabel et al. [9]	✓	✓	✓	✓	✓	✓		✓		general	point cloud	✓	
2009	Rusu et al. [2]	✓	✓	✓	✓	✓		✓	✓		tabletop	point cloud	✓	
2009	Rusu et al. [32]	✓	✓		✓		✓		✓	✓	tabletop	point cloud		
2010	Xiong et al. [35]	✓	✓					✓		✓	building floors	point cloud		
2011	Holz et al. [3]	✓	✓						✓		indoor	RGBD	✓	✓
2011	Li et al. [16]	✓	✓	✓	✓	✓		✓	✓		3d object	point cloud	✓	
2011	Koppula et al. [37]							✓		✓	indoor	point cloud, RGBD	✓	
2012	Nan et al. [21]									✓	indoor	point cloud		
2013	Xiong et al. [36]	✓	✓					✓		✓	building floors	point cloud		
2013	Ückermann et al. [4]	✓	✓						✓		tabletop	depth maps	✓	✓
2013	Huang et al. [34]	✓	✓		✓				✓	✓	industrial	point cloud		
2014	Oesau et al. [6]								✓		building floors	point cloud	✓	
2014	Xiao et al. [7]	✓	✓					✓			building floor	point cloud		
2015	Nguyen et al. [33]	✓	✓					✓			indoor	RGBD	✓	✓
2015	Tateno et al. [17]										indoor	depth maps	✓	✓
2016	Georgiev et al. [20]	✓		✓	✓	✓					objects on floor	point cloud	✓	✓
2016	Ochmann et al. [8]	✓	✓					✓	✓	✓	building floor	point cloud		
2016	Hackel et al. [22]									✓	outdoor	point cloud	✓	
2016	Armeni et al. [27]							✓		✓	building floors	point cloud	✓	
2016	Hettiarachchi et al. [12]	✓		✓	✓	✓	✓		✓		tabletop	point cloud	✓	✓
2017	Roberto et al. [18]	✓	✓	✓	✓				✓		general	point cloud	✓	✓
2017	Tchapmi et al. [25]									✓	general	point cloud	✓	
2018	Roberto et al. [19]	✓	✓	✓	✓				✓		general	point cloud	✓	✓
2017	Proposed approach	✓	✓	✓	✓				✓	✓	general	point cloud	✓	✓

[1] Plane model presence also refers to cuboids.
[2] Machine Learning is used (classification).
[3] A tick means relatively medium to high noise, whether no tick signifies relatively low noise, for example, as provided by a dense laser scan.
[4] Multiframe capabilities can indicate, but not necessarily imply, real-time performance.

Table 2.1: Point cloud segmentation methods overview.

3 Overview

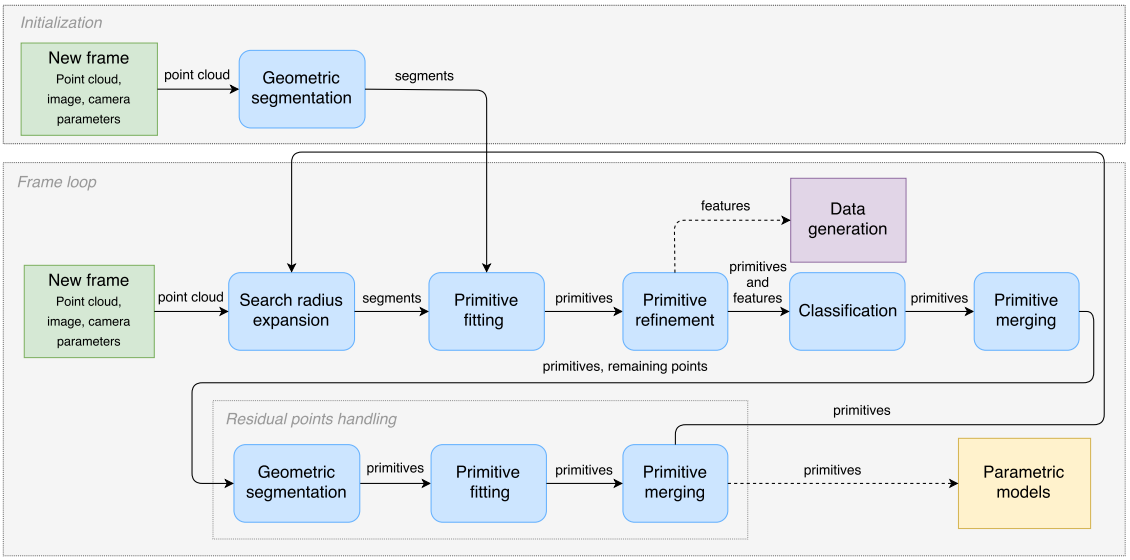


Figure 3.1: Overview of algorithm pipeline: the *initialization phase* takes place when processing the first frame. Afterwards the algorithm runs in the *frame loop* mode, which includes the *residual points handling* phase.

The goal of our method is to achieve a segmentation of a 3D scene into geometric primitives from a *stream* of frames. The models need to be improved over time and adapt to newly discovered geometry, as the acquisition device moves through the scene. This scenario is relevant when using an AR system relying on SLAM, where new parts of the scene are discovered opportunistically, and the user expects a quick system response.

While we do not rely on a consistent relationship between individual 3D points, we assume the point clouds are dense and already registered in a global coordinate system. The scenarios are static, due to the nature of the sensors, and using SLAM implies that a significant amount of noise is present. We assume indoor scenes consisting of objects that can be approximated with the geometric primitives supported in our system, but

otherwise make no further assumptions about the structure of the scene.

3.1 Denomination Conventions

In the following, we refer to the 3D point cloud, the related RGB image and the camera pose as a *frame*. We use the terms *cluster* or *segment* for a set of 3D points, and we assume a 3D representation for all object unless otherwise noted. A 3D primitive may be referred to as *primitive*, *shape* or *model*. The discussed thresholds are chosen for real datasets.

3.2 Overview

The proposed algorithm is shown in Figure 3.1. In the *initialization* phase, we segment the 3D points from the first frame based on the orientation of their normals. As a result, we obtain segments $C = \{c_1, ..., c_n\}$ which are coherent w.r.t. their normal orientation. We then apply a fitting phase, considering all types of 3D primitives for each segment, choosing only a single primitive which fits best. For this stage, we incorporate the number of inliers and the surface normals orientation, obtaining a set of primitives, $P = \{p_1, ..., p_n\}$. As this is the most computationally expensive part of our algorithm, a full point cloud segmentation and fitting is not repeated for the rest of the modeling process.

For all subsequent frames, the procedure is outlined in the *frame loop* in Figure 3.1. First, the segments are refined by defining a region of interest obtained by projecting convex hulls of the inliers of previously detected primitives onto the image plane. We use a modified camera pose to expand the search radius by moving the camera slightly backwards along its z-axis. This way, more 3D points are projected into the image plane. After obtaining the new segments, the inliers are reassigned, resulting in an updated set of primitives, P. More details are later described in Section 4.3.

The refinement of each primitive p_k is performed by means of non-linear optimization (see Section 4.4). Afterwards, based on features describing the quality of the fit, a set of SVMs is used to decide whether p_k represents an invalid or a valid fit. In the former case, the primitive is discarded, and its inliers are released to the set of unassigned 3D points. At the same time, primitives are merged together based on proximity and on the similarity of the parameters, further described in Section 4.5.

At this stage of the algorithm, there exists a subset of 3D points that does not belong to any primitive. These 3D points are again segmented, and primitives are fit to obtain a smaller set of new primitives P_{new}. They are merged with the set of already detected primitives, $P = P \cup P_{new}$; we refer to this step as *residual points handling* in Figure 3.1. More details can be found in Section 4.6.

4 Structural Modeling

This chapter presents in detail the methods used to perform structural modeling of a scene in the context of a stream of incoming frames. While the Sections 4.1, 4.2, 4.4, 4.5 and 4.6 refer to procedures applied on a single frame, Section 4.3 describes how the detected primitives are propagated from one frame to another.

4.1 Segmentation

Given a point cloud, we aim to segment it first into regions defining objects or object surfaces, so the primitives are fit in the obtained regions. Our approach performs a surface normal-based segmentation, making use of the fact that spatial edges delimit objects and surfaces. The idea of first segmenting the scene based on normals information as a prerequisite of fitting some model is used in related work [2, 21], while the general idea of a normal-based segmentation of 3D data is also similar to related approaches [3, 4, 17, 35–37].

The surface normals are calculated as follows: For each point, a plane is fit locally to its six closest neighbors. The normal on the plane is set as the normal in the respective point. The number of neighbors is set experimentally, according to the density and level of noise of the used point clouds. An alternative choice of neighbors would pick all the points in a certain radius around the given point. Either way, the neighborhood has to be neither too large nor too small. If the neighborhood is too large, it is equivalent to the assumption that the region is a smooth surface, whereas choosing a too small neighborhood would make the normal orientation too sensitive to noise [3].

We employ two similar segmentation methods presented in related literature. While the segmentation introduced by Tateno et al. [17] focuses on convex objects, the segmentation approached described by Ückermann et al. [4] partitions the scene into object surfaces delimited by any large variation in the normals' orientation.

4.1.1 Geometric Segmentation into Surfaces

This method is based on the approach proposed by Ückermann et al. [4], namely splitting the scene into surfaces and edges. It implies using a point cloud operator to threshold the cosine between neighboring normals.

$$\lambda_i = \mathbf{n_p} \cdot \mathbf{n_i} \tag{4.1}$$

$$\lambda_p = \min \lambda_i \tag{4.2}$$

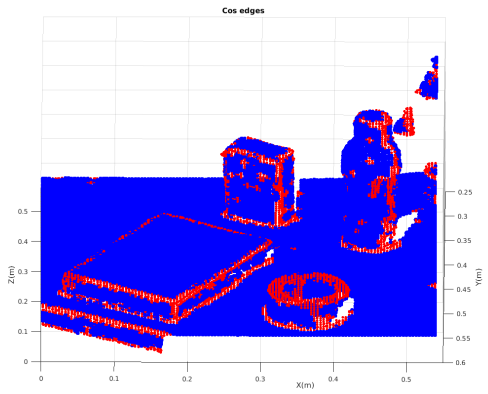
where $\mathbf{n}$ indicates a normal of a point, p is the index of the point where the operator is calculated, and i is the index of neighboring points within a radius r.

In contrast to the approach that inspired the segmentation method [4], we do not consider an 8-direction neighborhood for detecting the concavity, as we work directly with point clouds and not depth maps.

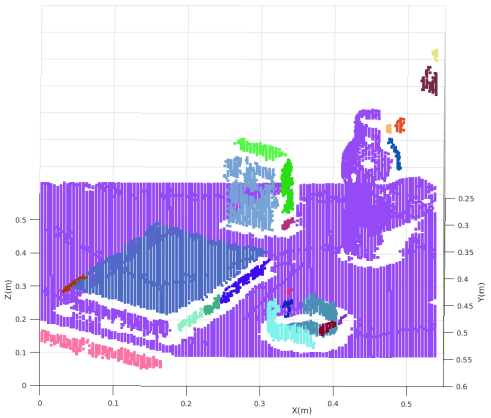
We choose the smallest angle between a normal in a point and the normals in the points within a radius r as edge criterion. If this angle is smaller than a threshold, we consider to have found an edge region. The threshold is set to to 0.8, which corresponds to around 36.9 degrees. We choose a low threshold, so that noise does not introduce edges erroneously [4]. The edges are removed, and the rest of the points are clustered. Clusters with fewer points than 0.04% of the whole point cloud are not considered further. The result of such a thresholding based on normal concavity is shown in Figure 4.1a.

After obtaining the edge and surface regions, we aim to retrieve the subsets of the point cloud that correspond to different object surfaces. The points marked as edges are removed from the cloud, and the Density Based Spatial Clustering of Applications with Noise (DBSCAN) [41] algorithm is used for clustering the remaining data. The algorithm works in a region-growing way: it starts with a seed point as belonging to a cluster and then it expands to neighboring points. A point located inside the cluster is called a core point, whereas a point situated on the edge of the cluster is called a border point. A core point has at least $MinPts$ neighbors in the ϵ radius around it in order to be a core point, and a border point can be in the neighborhood of such a core point. The algorithm starts with a random point, and expands the cluster based on the number of neighbors within a radius ϵ. When no points can be added anymore, another unused random point is chosen and the procedure is started again. The expansion can be performed only from core points. If the chosen point is not a core point, another point is picked instead.

There are two parameters to be set in the algorithm: the neighbor radius ϵ, and $MinPts$, the number of neighbors needed for a point to be considered a core point. We choose these parameters empirically, suited for the point clouds from our experiments. An example of obtained clusters is seen in Figure 4.1b.



(a) Output of point cloud edge detection from criterion 4.1. Points plotted in blue are clusters, while red points are cluster edges.



(b) Clusters obtained after running the DBSCAN algorithm on the point cloud after removing the identified spatial edges.

Figure 4.1: Segmentation of a point could into surfaces.

4.1.2 Geometric Segmentation into Convex Objects

As introduced by Tateno et al. [17], two operators are used to create a segmentation of a point cloud into convex objects. The first operator regards concave regions, and it is defined as follows:

$$\phi_i = \begin{cases} 1, & \text{if } (p_i - p) \cdot \mathbf{n_p} > 0 \\ \mathbf{n_p} \cdot \mathbf{n_i}, & \text{otherwise} \end{cases} \tag{4.3}$$

$$\phi_p = \min \phi_i \tag{4.4}$$

Here, p is the index of a point in the point cloud, i are the indices of the 9 nearest neighbors of the point with index p, and $\mathbf{n}$ indicates the normal in a point. After computing the value for each neighbor, the minimum is attributed to the point.

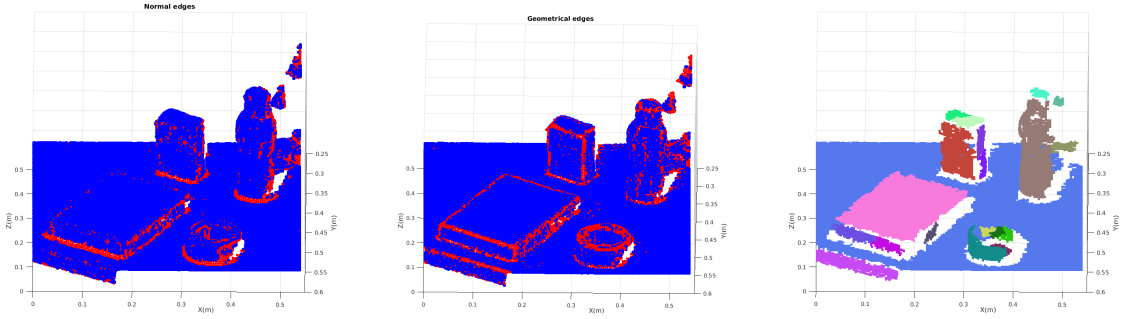
The second operator we consider when segmenting the point cloud is:

$$\Gamma_p = \max |(p_i - p) \cdot \mathbf{n_p}| \tag{4.5}$$

A threshold is applied to the values of the operators, using 0.96 for the first operator and 0.0013 for the second operator. The thresholds and the number of neighbors have been empirically chosen for the expected noise levels in the used dataset. The points where the ϕ operator is smaller than its threshold or the Γ operator is larger that the corresponding threhsold are considered edges in the point cloud. In Figures 4.2a and 4.2b, the outputs of the operators can be seen. The edges are removed and an undirected graph is built from

the point cloud based on point proximity. Afterwards, a connected component algorithm is applied on the graph, obtaining object segments. All segments with fewer points than 0.05% of the whole point cloud are not considered further, as they are likely just noise. If the cluster is supported by multiple objects, we assume that they will be detected in later frames, due to the strategy described in Section 4.6.

An example of the obtained clusters is depicted by Figure 4.2c, where each cluster is colorized differently. They are then used as input segments for the primitive fitting. This is described in Section 4.2.



(a) Output of point cloud edge detection from criterion described in Equations 4.3 and 4.4. Points plotted in blue are clusters, while red points are cluster edges.

(b) Output of point cloud edge detection from criterion 4.5.

(c) Point clusters obtained after a connected components algorithm on the point cloud after removing identified spatial edges.

Figure 4.2: Segmentation of a point could into convex objects.

4.2 Primitive Fitting

Three primitive types are fit to the clusters: plane, sphere and cylinder. After all three types are fit, the decision is taken regarding the best fitting primitive according to criteria described in detail in Section 4.2.5. The fitting is performed with RANSAC.

4.2.1 RANSAC

RANSAC is a robust iterative meta-algorithm for estimating parameters of mathematical models, when given data that contains noise. The main idea of RANSAC is that, by repeatedly picking as few samples as possible from a set of noisy data, there is a high chance that one chooses the right samples to correctly estimate the underlying model. It can be compared to a Least Squares approach: while Least Squares uses all available data for an estimate to even out the noise, RANSAC does exactly the opposite [42]. The algorithm can be briefly described by the following steps:

1. *Randomly choose the smallest possible set of data needed to be able to estimate the desired model.*
2. *Estimate the model from the chosen subset and check how many samples support this model, i.e. how many samples are within a distance d from the model; these samples are the consensus set.*
3. *If the consensus set is large enough, this model is taken into consideration.*
4. *Repeat steps 1 - 4, until a model with enough inliers t has been found, or maximum number of iterations k is reached.*
5. *Reestimate the model using all the inliers.*

RANSAC uses the following parameters: d - the error threshold to decide whether a sample is an inlier, t - the number of inliers that suffice to state that the model has been found, and k - the maximum number of iterations allowed.

The parameter d is set empirically, based on the scale of the scene. The parameter t is not needed in our approach, as the variant of RANSAC that we use does not include it. The maximum number of allowed iterations k is set to 1000, not restricting the algorithm, as the expected number of iterations for the algorithm to find the correct model with 99.99% certainty, assuming 70% inliers is 22 for planes, 34 for spheres, 14 for cylinders. This is calculated following the formula [13]:

$$k = \frac{log(1-z)}{log(1-w^n)} \tag{4.6}$$

Here, z is the probability that the correct model was found, w is the ratio of inliers to the total number of points, and n is the number of points one needs to sample for estimating the model.

M-estimator Sampling and Consensus (MSAC) [43] is an improved variant of RANSAC implemented by MATLAB, which we use for performing primitive fitting. RANSAC minimizes the following objective function with its associated per-point cost [43]:

$$C = \sum_i cost(error_i^2) \tag{4.7}$$

$$cost(error^2) = \begin{cases} 0, & \text{if } error^2 < \theta \\ c, & \text{otherwise} \end{cases} \tag{4.8}$$

Here, c is a constant and θ is a threshold. In contrast to this, MSAC uses the following cost function per point:

$$cost(error^2) = \begin{cases} error^2, & \text{if } error^2 < \theta \\ \theta, & \text{otherwise} \end{cases} \tag{4.9}$$

Therefore, while RANSAC penalizes each outlier the same way, MSAC also accounts for how precisely the inliers fit the model, using a modified cost function that accumulates the cost for each inlier, while keeping a constant penalty for the outliers.

The complexity of the two algorithm variants is the same. As pointed out by Torr et al. [43], both MSAC and Maximum Likelihood Consensus (MLESAC) outperform RANSAC. MLESAC, a Maximum Likelihood Estimation (MLE) variant of RANSAC, outperforms MSAC, but requires more computations. Therefore, in our case, where we aim to run the algorithm each frame, choosing MSAC is a good trade-off between performance and efficiency.

RANSAC cannot deal with finding more than one model when more underlying models are present in the dataset, but, in our case, this problem is avoided in two ways. First of all, the scene is segmented before the fit into clusters with coherent normal orientation, as previously described. Secondly, the points that were not assigned to any model are segmented again, and there is another attempt to fit primitives to them. This is presented in Section 4.6.

Step 2 of the RANSAC algorithm, regarding the model estimation, is described in the following subsections with details about estimating the parameters of the geometric models.

4.2.2 Plane Model Fitting

A 3D plane is characterized by its orientation, or normal, and by its positioning in 3D space, represented by a distance from the origin. The general form of the plane equation is:

$$ax + by + cz + d = 0 \tag{4.10}$$

Here, the normal $\mathbf{n} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$, $-d$ is the distance from the origin to the plane, and x, y, z are the coordinates of a point that satisfies the plane equation. The set of parameters defining a plane are:

$$P_{plane} = \{a, b, c, d\} \tag{4.11}$$

In the RANSAC loop, the plane is estimated from three points: $\mathbf{p_1}$, $\mathbf{p_2}$, $\mathbf{p_3}$. The normal on the plane and the parameter d are calculated as:

$$\mathbf{n} = ||(\mathbf{p_2} - \mathbf{p_1}) \times (\mathbf{p_3} - \mathbf{p_1})|| \tag{4.12}$$

$$d = -\mathbf{p_1} \cdot \mathbf{n} \tag{4.13}$$

4.2.3 Sphere Model Fitting

The general form of the sphere equation is:

$$(x-a)^2 + (y-b)^2 + (z-c)^2 - r^2 = 0 \tag{4.14}$$

Here, a, b and c are the coordinates of the center of the sphere, r is the radius, and x, y, and z are the coordinates of a point lying on the sphere. The set of parameters defining a sphere are:

$$P_{sphere} = \{a, b, c, r\} \tag{4.15}$$

These are the parameters that have to be estimated. In the RANSAC loop, the sphere is estimated from four points, p_1, p_2, p_3, p_4, by solving a linear system of equation with four equations and four unknowns using Cramer's rule:

$$a = -\frac{D_a}{2D} \quad b = -\frac{D_b}{2D} \quad c = -\frac{D_c}{2D} \tag{4.16}$$

$$r = \sqrt{a^2 + b^2 + c^2 - 4 \cdot \frac{D_r}{D}} \tag{4.17}$$

where

$$D = \begin{vmatrix} p_{1_x} & p_{1_y} & p_{1_z} & 1 \\ p_{2_x} & p_{2_y} & p_{2_z} & 1 \\ p_{3_x} & p_{3_y} & p_{3_z} & 1 \\ p_{4_x} & p_{4_y} & p_{4_z} & 1 \end{vmatrix} \tag{4.18}$$

$$D_a = \begin{vmatrix} d_1 & p_{1_y} & p_{1_z} & 1 \\ d_2 & p_{2_y} & p_{2_z} & 1 \\ d_3 & p_{3_y} & p_{3_z} & 1 \\ d_4 & p_{4_y} & p_{4_z} & 1 \end{vmatrix} \qquad D_b = \begin{vmatrix} p_{1_x} & d_1 & p_{1_z} & 1 \\ p_{2_x} & d_2 & p_{2_z} & 1 \\ p_{3_x} & d_3 & p_{3_z} & 1 \\ p_{4_x} & d_4 & p_{4_z} & 1 \end{vmatrix} \tag{4.19}$$

$$D_c = \begin{vmatrix} p_{1_x} & p_{1_y} & d_1 & 1 \\ p_{2_x} & p_{2_y} & d_2 & 1 \\ p_{3_x} & p_{3_y} & d_3 & 1 \\ p_{4_x} & p_{4_y} & d_4 & 1 \end{vmatrix} \qquad D_r = \begin{vmatrix} p_{1_x} & p_{1_y} & p_{1_z} & d_1 \\ p_{2_x} & p_{2_y} & p_{2_z} & d_2 \\ p_{3_x} & p_{3_y} & p_{3_z} & d_3 \\ p_{4_x} & p_{4_y} & p_{4_z} & d_4 \end{vmatrix} \tag{4.20}$$

$$\begin{aligned} d_1 &= -(p_{1_x}^2 + p_{1_y}^2 + p_{1_z}^2) \\ d_2 &= -(p_{2_x}^2 + p_{2_y}^2 + p_{2_z}^2) \\ d_3 &= -(p_{3_x}^2 + p_{3_y}^2 + p_{3_z}^2) \\ d_4 &= -(p_{4_x}^2 + p_{4_y}^2 + p_{4_z}^2) \end{aligned} \tag{4.21}$$

4.2.4 Cylinder Model Fitting

As there is no closed form equation for a finite cylinder, we fit an infinite cylinder and then define the caps.

The estimation is performed as follows: two randomly sampled points are chosen to be candidates on the surface of the cylinder. The closest distance between the normals of the two points is calculated, and the intersection of the two normals is considered to be on the cylinder axis. If the normals do not intersect, the point in the middle of the shortest connecting segment is chosen. The orientation of the axis is calculated as the cross product of the two normals. The radius is the mean distance from the surface points to the axis point. Six neighbor points are used to calculate the surface normals for each point. The calculation of the normals in described in Section 4.1.

The points situated on the surface of the cylinder caps are not included into the cylinder fitting procedure. Therefore, they are neither counted in in the RANSAC loop as inliers, nor do they add up their cost from Equation 4.9 during the minimization. Thus, they have no influence on the resulting cylinder. In other words, tubes are fit.

An estimated cylinder is defined by the axis $\mathbf{a} = \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix}$, a point on the axis $\mathbf{p} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}$ and the radius r:

$$P_{infinite\_cylinder} = \{a_x, a_y, a_z, p_x, p_y, p_z, r\} \tag{4.22}$$

The height of the infinite cylinder is limited by projecting all its inliers on the cylinder axis and choosing the axis limits as the minimum and maximum of the inlier projections. A cylinder model is obtained that is defined by the the center of the first cylinder cap $\mathbf{c1} = \begin{pmatrix} c1_x \\ c1_y \\ c1_z \end{pmatrix}$, the center of the second cylinder cap $\mathbf{c2} = \begin{pmatrix} c2_x \\ c2_y \\ c2_z \end{pmatrix}$ and the radius r.

$$P_{cylinder} = \{c1_x,\ c1_y,\ c1_z,\ c2_x,\ c2_y,\ c2_z,\ r\} \tag{4.23}$$

4.2.5 Fitting Decision

In order to decide which primitive is fit, three criteria are taken into consideration:

- the maximal allowed radius, in the case of cylinder and spheres. This is described in Section 4.2.5.1.
- number of inliers where the surface normals do not deviate from the direction of the normals on the surface of the fit primitive within a threshold. This criterion is further described in Section 4.2.5.2.
- the uniformity of the distribution of the inliers on the primitive surface. This criterion is described in more detail in Section 4.2.5.3.

4.2.5.1 Size Restriction on Spheres and Cylinders

When noise is present, each plane can be approximated by a large enough sphere or cylinder. Such a case is shown in Figure 4.3b. In order to avoid fitting very large such primitives to planes, we restrict the allowed size of the spheres and cylinders. This also means that we assume that we cannot have spheres and cylinders with the diameter larger than 80% of the width, depth and height of the bounding box of the scene. The primitives that are larger than this allowed size are set as invalid fits. A similar criterion is used by Roberto et al. [19].

4.2.5.2 Normal Curvature

An additional criterion for a good fit, also used by Schnabel et al. [9], is the deviation of the point cloud surface normals in comparison to the normals of the fit primitive. From the inliers delivered by RANSAC, only the inliers that respect this criterion are taken into consideration for choosing the best primitive for a cluster.

The cosine of the angle between normals is calculated as follows:

$$\psi_i(\mathbf{n_f^i}, \mathbf{n_p^i}) = \begin{cases} \mathbf{n_f^i} \cdot \mathbf{n_p^i}, & \text{if } \mathbf{n_f^i} \cdot \mathbf{n_p^i} \geq 0 \\ \mathbf{n_f^i} \cdot -\mathbf{n_p^i}, & \text{otherwise} \end{cases} \tag{4.24}$$

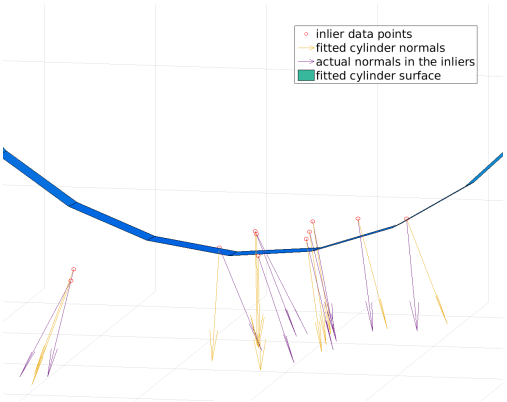
Here, $\mathbf{n_f^i}$ is the normal on the surface of the fit primitive at point i, while $\mathbf{n_p^i}$ is the normal corresponding to point p_i based on its neighbors in the point cloud. If the calculated angle is larger than 180 degrees, one of the normals is flipped. An example showing the normals on a fit cylinder is displayed in Figure 4.3a.

In our case, the inliers are not accurate regarding the inside and outside of a surface, as the underlying shapes are unknown, and the point clouds are captured by multiple poses and merged together inside of the sensor. Therefore, in case the cosine is negative, or, in other words, the angle is obtuse, it means that one normal is pointing inside and another one outside. In this case, we flip one normal and recalculate the cosine. We threshold this deviation angle, by allowing a maximum of 18 degrees deviation, to account for noise in the cloud [4].

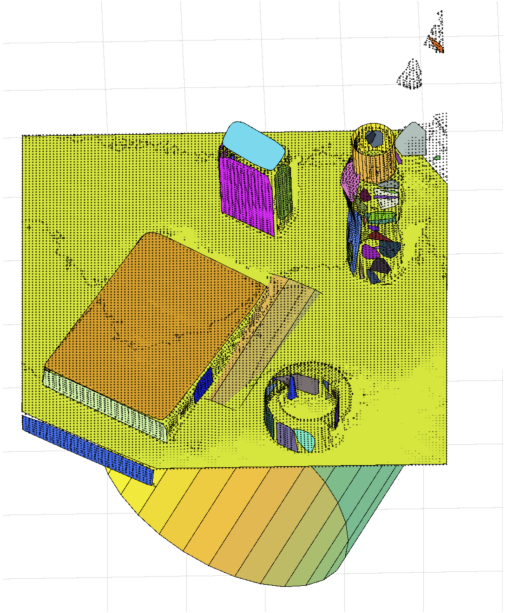
4.2.5.3 Inlier Distribution on the Primitive Surface

Another measure to discard cylinders and spheres that approximate noisy planar surfaces is considering the surface inliers distribution based on the observation that, in the situation of a wrong fit, the inliers are distributed in a highly non-uniform way on the primitive surface. We propose to quantify the similarity of the distribution of the inliers on the cylinder or sphere surface to a uniform distribution.

As a similarity measure, we use the Hellinger distance. The Hellinger distance is defined for discrete distributions as:



(a) Normals on the point cloud surface (purple) and normals on the fit cylinder surface (yellow); the angle between them is used as a criterion to further filter inliers.



(b) Large cylinder approximating a noisy region of a horizontal plane.

Figure 4.3: The influence of the deviation of the point cloud surface normals from the primitive surface normals when fitting a cylinder.

$$H(P,Q) = \frac{1}{\sqrt{2}} \sqrt{\sum_{i=1}^{n} (\sqrt{p_i} - \sqrt{q_i})^2} \tag{4.25}$$

Here, P and Q are the two discrete distributions, where p_i and q_i are the probabilities of the outcome with index i. Intuitively, it can be understood as the Euclidean distance between the square roots of multidimensional distribution vectors; the distance is maximal when one distribution has zero probabilities while the other distribution has positive probabilities.

Two distributions need to be determined: the surface distribution P and the ideal uniform distribution Q. To calculate the surface probabilities, we discretize the areas, by splitting each shape surface into a number of surface bins. The probability of a point being in bin i is calculated as follows:

$$p_i = \frac{p_{b_i}}{n} \tag{4.26}$$

$$q_i = \frac{\frac{n}{n_b}}{n} = \frac{1}{n_b} \tag{4.27}$$

Here, p_{b_i} is the number of points in bin i on the primitive surface, n is the total number of points on the surface, and n_b is the number of bins. The bins are defined as follows, for each primitive type:

Plane
The plane and its inliers are rotated to be parallel with the horizontal plane. Then, it is split into a 2D grid of 8×8 bins.

Sphere
The 3D Euclidean coordinates of the inliers are transformed into spherical coordinates, defined by azimuth, elevation and radius. As the radius does influence the distribution on the surface and as the inliers are very close to the surface, we take into account the azimuth and elevation and calculate a 2D histogram, split into 6×6 bins.

Cylinder
In the case of a cylinder, a similar procedure is performed. The cylinder is rotated to vertical position and brought to the origin. The 3D Euclidean coordinates of the cylinder inliers are transformed into cylindrical coordinates, defined by angular, radial and elevation coordinates. The radius is ignored, as we assume the inliers are close to the cylinder surface, and the other two coordinates are not influenced by the radius. We calculate the 8×2 2D histogram of the angular and elevation coordinates.

The reason to choose more bins when considering the angular coordinates, is that the most representative indicator of a good cylinder fit consists of the distribution of the points around the axis and not along the axis.

For each shape, we calculate the discrete distribution P by counting the number of inliers per bin and dividing by the total number of inliers, as described in Equation 4.26. The discrete distribution is compared to a uniform distribution in terms of the Hellinger distance, obtaining an indicator of uniformity of the distribution of the points on the surface. The chosen uniformity threshold is very tolerant, as just extreme cases need to be excluded, where the points are clustered together tightly in isolated clusters on the surface; a threshold of 0.7 is chosen, considering that possible values of the distance lay in the interval [0, 1].

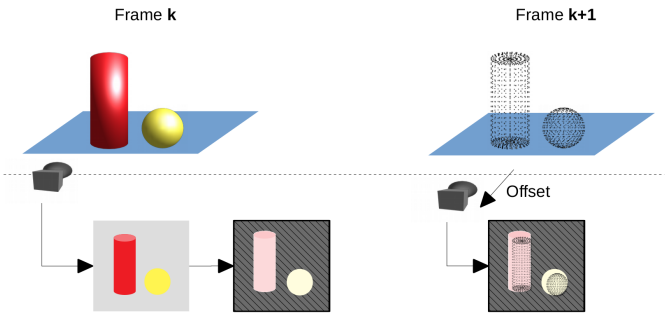
4.3 Label Propagation and Search Radius Expansion

When a new frame needs to be processed, there is no correspondence between the inliers of a shape in a given frame k, and its inliers in the next frame $k+1$. A strategy is needed for propagating the labels in the point clouds between frames and also for expanding the shapes by adding new inliers. For each primitive s_i, where i is the index of the primitive in its frame, the following steps are performed:

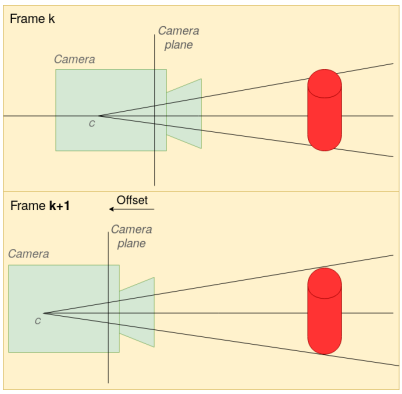
- A new set of camera parameters for frame $k+1$ is calculated by moving the camera center backwards by a fixed offset along the vector that is connecting the camera center to the centroid of the inliers of the shape s_i; we obtain the modified projection matrix $P'_{i+1} = K_{i+1} \cdot R(I| - C')$, where K is the camera intrinsics matrix, R is the rotation matrix, and C' is the modified camera center.

- We project the convex hull of the set of inliers of primitive s on the image plane of frame $k+1$ using P'_{i+1}. This has the effect that the search cone for finding new inlier candidates is increased, as if the camera would be able to see more around the inliers. Figure 4.4b shows this effect in a simplified scenario.

- All the points that lie in this new convex hull are considered inlier candidates. A check is performed to decide which of the new inlier candidates fit to primitive s, according to its inlier criterion.



(a) Label propagation between frame k and frame $k+1$, by projecting the inliers of each primitive onto the camera plane corresponding to a modified pose obtained by moving the camera backwards by a fixed offset.



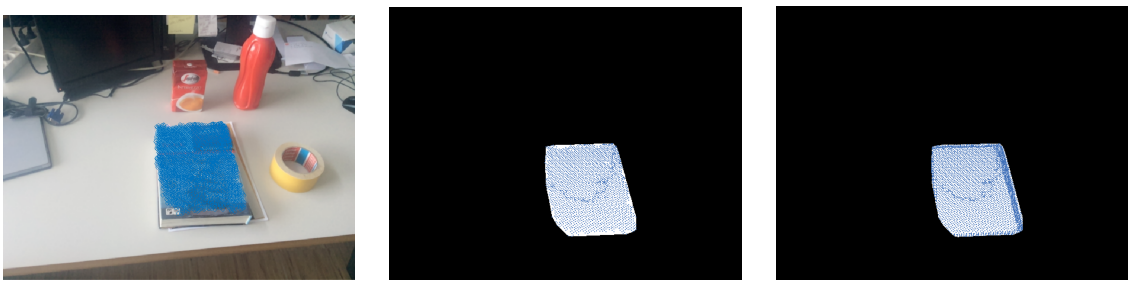
(b) Simplified scenario to show the effect of the size of a fixed object in the image caused by moving the camera backwards.

Figure 4.4: Propagating primitive labels and expanding inliers search region between two consecutive frames.

The procedure is depicted in Figure 4.4 and 4.5. This strategy defines a search cone for each primitive in the next frame. The objects that are behind or in front of the considered primitive are not assigned to the primitive, as they do not pass the inlier criterion, which is taking into account the spatial proximity to the model.

A similar idea based on label projection on the image plane to propagate the object labels between frames is proposed by Tateno et al. [17].

This approach works because a static scene is assumed; both the camera and the objects have little movement between the frames, therefore, little optical flow. By expanding the search regions of the inliers, the support of the primitive is also potentially increased.



(a) (b) (c)

Figure 4.5: Expanding the search region of a plane's inlier candidates in the next frame. The inliers of the book cover (left) are projected onto the image plane with modified camera matrix P', creating a region growing effect (right). The middle image shows the mask of the projected inliers using the original projection matrix P.

When there is a plane in the scene that is perpendicular on the camera plane, the projected inliers are collinear. The convex hull of the cluster cannot be calculated. In such a case, the projected convex hull needs to be artificially expanded. If the number of inliers is small, and the convex hull cannot be calculated because of collinearity in the projection, the projected binary mask of the cluster is dilated with a 3×3 all-ones structuring element. Then the convex hull of this expanded mask is calculated, obtaining a search window for inlier candidates in frame $k+1$. A similar procedure is proposed by Nguyen et al. [33], where single-pixel paths of a segmented cluster in the image plane are increased by adding surrounding pixels.

4.4 Primitive Refinement

After the parameters of the primitives have been estimated, a refinement step follows performing non-linear optimization to improve the fit. Rusu et al. [2] also use a non-linear optimization after RANSAC primitive fitting. This optimization step accounts for nonlinearities caused by sensor distortions or noise. Using the Ceres solver [39], we initialize the optimization process with the already estimated primitive parameters, and optimize iteratively using the Levenberg-Marquardt algorithm. The Ceres solver offers the option to automatically calculate gradients, which we choose in our implementation. Dense QR decomposition is used as a linear solver.

For each shape, the cost function for a 3D point $\mathbf{p}$ is defined as follows:

Plane

$$cost_{plane} = \left| \frac{\mathbf{n} \cdot \mathbf{p} + d}{||\mathbf{n}||} \right| \tag{4.28}$$

The variables are defined in Section 4.2.2, as well as the set of parameters that we optimize, P_{plane}.

Sphere

$$cost_{sphere} = |||\mathbf{c} - \mathbf{p}|| - r| \tag{4.29}$$

c is the center of the sphere, **p** is a point on the sphere and r is the length of the radius of the sphere. The set of parameters that we optimize is P_{sphere}, as described in Section 4.2.3.

Cylinder

Having a cylinder defined by its axis, a point on the axis and its radius, we define the cost function as the distance from the point to the surface of the cylinder. This distance is obtained by first calculating the distance from the point to the cylinder axis, by using the fact that the cross product of two vectors is the area of the parallelogram spanned by the vectors. By dividing the parallelogram area by half, the area of a triangle is obtained. As the triangle is spanned by the cylinder axis vector and the distance from the point to the cylinder axis, the distance can be calculated by using the formula for the area of a triangle. Finally, the radius is subtracted to get the distance from the point to the cylinder surface.

$$cost_p = \left| \frac{\mathbf{axis}' \times (\mathbf{p_{axis}} - \mathbf{p})}{||\mathbf{axis}||} - r \right| \tag{4.30}$$

$\mathbf{p_{axis}}$ is a point on the cylinder axis, and **p** is a point on the surface. The set of parameters that we optimize is $P_{cylinder}$, as described in Section 4.2.4.

To make the fitting more robust to outliers, a loss function is used, which is provided by the Ceres solver. We choose the Huber loss function, and the Ceres solver implements this variant:

$$\rho(r) = \begin{cases} r^2, & \text{if } |r| \leq \alpha \\ 2 \cdot \alpha \cdot |r| - \alpha^2, & \text{otherwise} \end{cases} \tag{4.31}$$

Here, r is the residual, and α is a threshold for defining an outlier in terms of distance from the sample to the model. In this way, the fitting converges to the actual underlying shape.

The effect of this function on top of the cost function is that the points with a large cost, which are likely to be outliers, influence the fit less than the points that are closer to the shape, making the measure more robust. The plot of a Huber loss function applied on a convex parabola can be seen in Figure 4.6.

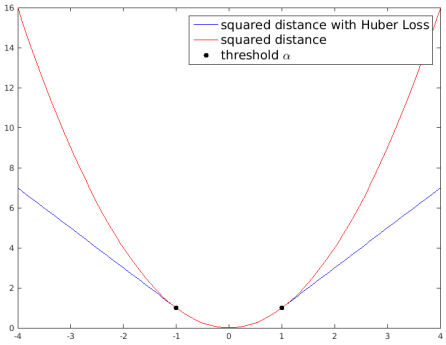


Figure 4.6: Quadratic function and the Huber loss function applied on the quadratic function, with threshold $\alpha = 1$.

4.5 Primitive Merging

When primitives have similar parameters and are close to each other, we assume that they describe the same primitive, so a merging strategy is needed. The merging is necessary if the geometric segmentation led to over-segmentation, or an object was partly occluded by other objects and, in later frames, the updated geometry reveals that previously isolated parts belong to the same object. The merging strategy is different depending on the primitive type. The thresholds used for checking shape similarities are empirically chosen and scaled according to the nature of the point clouds.

Plane

Plane primitives are fit, resulting in plane segments. We define plane segments as convex hulls of plane inliers projected on the plane [3, 16, 35]. A discussion about the use of convex hulls for representing planes is included in Section 7.1.

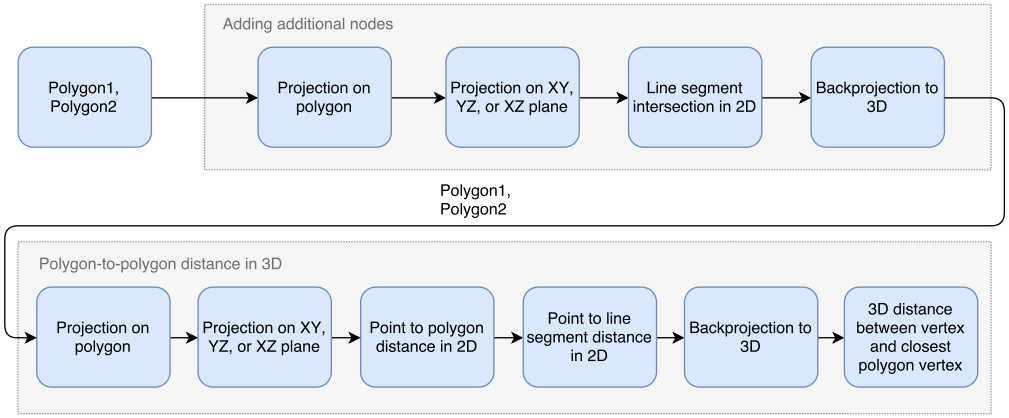


Figure 4.7: The steps required for computing the minimal polygon-to-polygon distance in 3D.

In order to merge two plane segments, more checks have to be performed. One of the checks is that the angle between the normals of the two plane models has to be smaller than a threshold, which we set to approximately 10 degrees. Next, the plane segments have to be close to each other. As we are dealing with finite plane segments, it is not sufficient to compare the parameters of the planes, because the proximity of the plane segments depends on the spatial location of the plane segments on the planes. Thus, the minimal distance between the two plane segments has to be computed as the minimal distance between two polygons in 3D. The steps for calculating the polygon-to-polygon 3D distance are also shown in Figure 4.7.

First, it is checked whether the polygons intersect in 3D; if they intersect, the distance between them is zero. This check is performed by intersecting each edge of a polygon with the plane of the other polygon, and then checking whether the intersection is inside the second polygon. As soon as such an intersection point is found, the two polygons intersect.

If the polygons do not intersect, the polygon-to-polygon distance is calculated. Different situations of positions of triangles are shown in Figure 4.8.

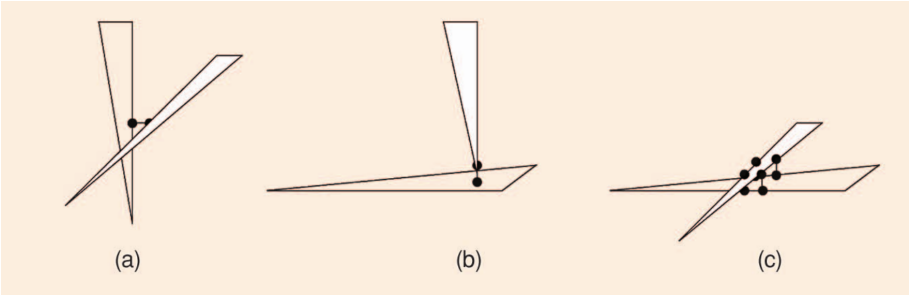
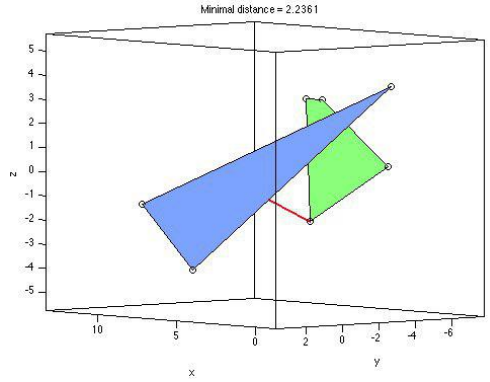


Figure 4.8: Different situations of minimal distances between triangles in 3D [1].

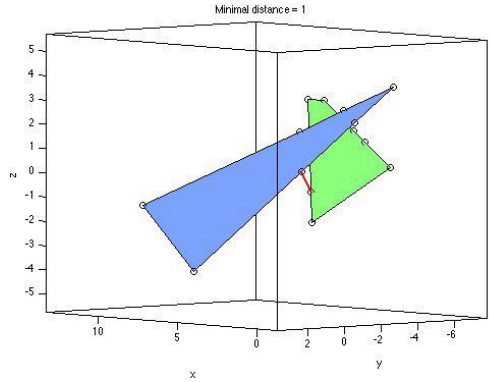
The polygon-to-polygon distance is based on the idea of computing the minimum of the the minimal distances from each vertex of a each polygon to the other polygon. However, there are also cases when the minimal distance between two polygons cannot be calculated as a point-to-polygon distance. Such a situation can be seen in Figure 4.8 and in Figure 4.9. In this case, the minimal distance is an edge-to-edge distance. Because of this, we introduce new vertices in the polygons, by projecting them on each other and calculating the intersection of their edges in 2D. The intersections are then projected back on to the polygons in 3D, resulting in new vertices on the convex hull.

After this step, the discussed vertex-to-polygon minimal distance can be computed. This is implemented based on the point-to-polygon 3D distance [44]: Given a point and a polygon, the point is projected onto the plane of the polygon, and they are both projected on one of the orthogonal planes XY, XZ, or YZ, based on the least variation of the points in the plane, indicated by the maximum of the three components of the normal vector. Now the problem has been reduced to a 2D point-to-polygon minimal distance. This can be solved by calculating the minimal distance between the point and each edge of the

polygon. Once the points on the polygon or within the polygon that yield the minimal distance are found, they are projected back to 3D on the polygon's plane. The point-to-polygon minimal distance in 3D is found by taking the 3D distance between these back projected points and the original 3D point.



(a) Distance between 2 convex polygons in 3D before introducing vertexes by intersecting of the 2D projections of the polygons on each other.



(b) Distance between 2 convex polygons in 3D after introducing vertexes by intersecting of the 2D projections of the polygons on each other.

Figure 4.9: Introducing new vertices in a polygon for calculating the minimal polygon-to-polygon distance in 3D.

The inliers of the new shape are set as the union of the inliers of both shapes. In the refinement step, all inliers will be taken into consideration for adjusting the plane parameters, therefore, the shape will be updated with respect to all inlier candidates. The new convex hull is the convex hull of the points defining the two convex hulls. The parameters of the plane are the parameters of the plane with the most inliers.

Sphere

In the case of sphere merging, the parameters of the two spheres are compared. If the distance between the sphere centers and the difference between the radii are below accepted thresholds, the two spheres are merged. The parameters of the resulting sphere are the the ones of the sphere with the most inliers, and the inliers are the union of the two inlier sets.

Cylinder

For cylinders, first, the angle between the axes is checked. If this angle and the difference between the radii are smaller than a threshold, the minimal distance between the cylinder axes is calculated. It is assumed that the cylinders have the same parameters and further checks are necessary regarding a cylinder intersection.

More situations for cylinder intersections can be distinguished: The cylinders can either

overlap each other partly, or one can be contained in the other. These cases are dealt with by projecting their caps on one axis, and then considering the 1D line segment overlap. Thus, if the cylinders intersect, they are merged and the obtained cylinder receives the parameters of the cylinder with the most inliers, except the coordinates of the caps. These are computed in the same way as when a new cylinder is fit. The inliers are in this case also the union of the inliers of both shapes.

4.6 Handling Remaining Points

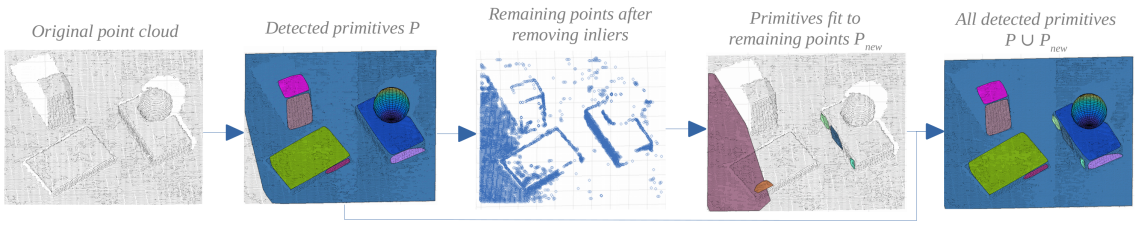


Figure 4.10: The workflow of the algorithm integrating the new primitives detected by handling the remaining points: Primitives P are fit to the point cloud. The primitive inliers are removed from the original point cloud. New primitives P_{new} are fit to the remaining points; they are eventually merged with the already detected primitives.

After computing a set of primitives $P = \{p_1, ..., p_n\}$ for a frame k, there are some points that are still unassigned. The points are left from previous steps in the algorithm, such as the initial segmentation, where they were identified as edges, or they belonged to clusters that were considered too small. Alternatively, they were discarded as outliers in the model fitting or model refinement steps. The reasons for this are either because they were considered noise, or because the segmentation was not granular enough so that the cluster that the points belong to described more than one primitive. As RANSAC can only fit one model at a time, just part of the points were assigned. However, most of the remaining points come from the classification step, where the primitive they had been assigned to was discarded. For these points, new primitives have to be fit.

The remaining points are therefore segmented into clusters like in the initial segmentation described in Section 4.1, but in this case, the segmentation criterion described in Section 4.1.1 is used, obtaining a finer segmentation. Thereby, smaller objects and more subtle details are modeled as well.

Primitives are fit to the resulting clusters, obtaining a new set of shapes P_{new} that are either merged with existing primitives, or added in the global set of primitives. This step makes sure that all points are used, and that new primitives come in, revealing new or updated geometry.

Figure 4.10 shows an outline of this step. In the algorithm pipeline, this procedure is described as the *residual point handling*, depicted in Figure 3.1.

5 Classification

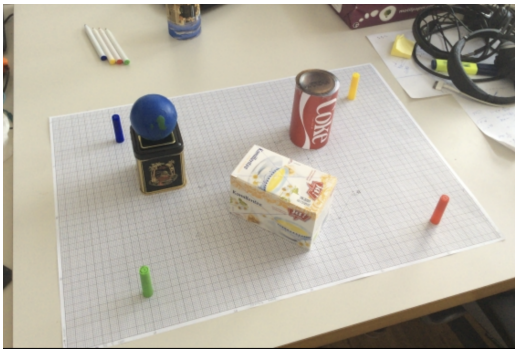
During acquisition, objects are only partially visible to the sensor. In certain cases, especially at the beginning of a scanning session, this leads to the detection of wrong primitives, as only little data is available. Therefore, a criterion is needed to decide when to discard these misfits at a later stage, such that a refitting step can be rerun in this particular region.

Deep learning would likely be able to handle such a challenging decision; recent work in 3D semantic segmentation with neural networks is showing promising results [25, 30]. However, training requires vast amounts of largely noise-free training data, which, in our case, is exhausting to acquire, as public datasets are not available. We therefore decided in favor of SVM as a more traditional classification technique, which can be applied on a standard CPU with manageable training effort. We use three binary SVMs that, given five features describing the fit of a primitive, evaluate whether it should be kept or discarded. The classification is run every three frames on the set of detected primitives, before the *handling remaining points* step.

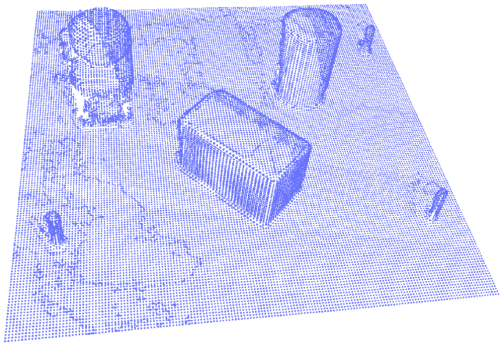
5.1 Training Data

As there are no publicly available labeled primitive datasets, we generated our own. After experimenting with training on artificial labeled datasets, we found that using the classifiers on real-world data sets yields a rather poor performance. This is due to the nature of the noise that is not modeled accurately by the artificial data by assuming normal or uniform distributions. Therefore further investigation into the nature of the noise is needed.

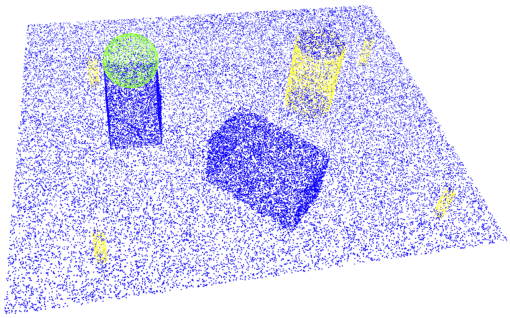
Because of this, labeled real-world datasets are created: First, objects are placed on grid paper so that their exact position is known. Four 3D targets are placed at the edges of the scene for scaling purposes. Then the scene is recorded with the sensor and the object setup is recreated in Blender and sampled to obtain point clouds. The two dense point



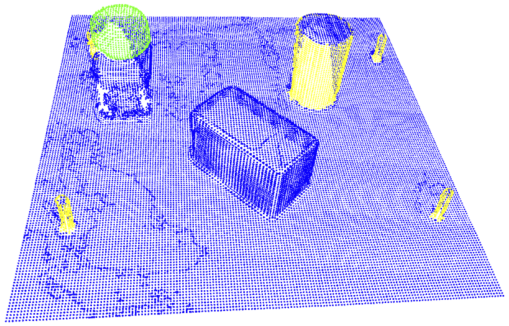
(a) Image of objects placed on grid paper.



(b) Captured point cloud.



(c) labeled artificial point cloud built as a replica of the real point cloud.



(d) Real point cloud with transferred labels.

Figure 5.1: Transferring primitive labels from the artificial point cloud (replica of the real data) to the real-world point cloud.

clouds are scaled and then aligned with Iterative Closest Point (ICP), considering only the four targets. The Blender scene is color-coded, meaning that each point has a color that indicates the type of shape it belongs to. These labels are transferred by calculating the nearest neighbor of each point from the real-world point cloud in the artificial point cloud. The result is a labeled real-world point cloud, as depicted in Figure 5.1.

This procedure requires a lot of effort, hence, the small datasets for training and testing.

5.2 Features

Each model fit is described by five signature features, which capture geometric properties of the primitive and its inliers. The features are defined as follows:

1. **#inliers**: the number of points that are located within a tolerated distance to the fit primitive. Primitives with over 10,000 inliers are not classified at all; they are considered reliable.

2. **rmse**: the root mean squared error of the distance from all the inliers to the fit primitive.

3. **#inliers_nde**: the number of inliers that agree with the direction of the normals on the surface of the shape. This criterion is further described in Section 4.2.5.2.

4. **mnde**: the mean error, in cosine of angles, of the deviation of the surface normals from the normals on the primitive, also described in Section 4.2.5.2.

5. **hellinger_dist**: the uniformity of the distribution of the inliers on the primitive surface. This criterion is described in more details in 4.2.5.3.

A sample s is a 5-dimensional vector of the form:
$s = \{\#inliers,\ rmse,\ \#inliers\_nde,\ nde,\ hellinger\_dist\}$
Each feature is scaled to [0, 1] by dividing by its maximum value.

5.3 Samples Generation

The samples are generated by running the algorithm on real-world labeled datasets and identifying correct and incorrect fits. The runs are carried out in cycles of two frames, so that every two frames a segmentation of the whole point cloud takes place. Without update over multiple frames, samples are less interdependent.

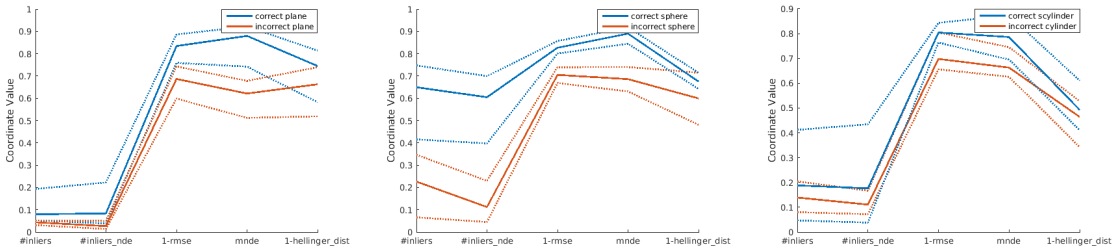
The algorithm is run three times, every time force-fitting a primitive type, in order to acquire positive and negative samples of the respective type.

At a given frame k, a sample $s^k_{p_i}$ is generated for each detected primitive p^k_i. The class label of the sample is computed as follows: the real type of the primitive is deduced from the labeled inliers, as the type of the majority of the inliers, indicated by their color [35]. The real primitive type is compared to the primitive type of p^k_i. If they are the same, then the class label of the sample is set to 1, otherwise, 0.

Plots of the median values with lower and upper quartiles of the generated samples are shown in Figure 5.2, where the separability of the classes can also be seen. By inspecting this visualization, we expect that the sphere is the easiest to classify, because the $\#inliers$ and the $\#inliers\_nde$ features strongly indicate the class of the samples.

5.4 Training and Testing

For each primitive type, and each kernel type, we perform 5-fold cross-validation and train a binary SVM. We experimented with radial basis function kernels, third degree polynomial kernels and linear SVM.



(a) Median of plane features. **(b)** Median of sphere features. **(c)** Median of cylinder features.

Figure 5.2: Parallel coordinates plot of feature values for each primitive type, with median (continuous line) and upper and lower quartiles (dotted line) per class.

Moreover, we also repeat this procedure for feature selection to find the best combination of features for a particular primitive type. The following configurations of features were evaluated:

$$\{1,2,3,4,5\} \quad \{3,4,5\} \quad \{1,3,4,5\} \quad \{1,3,4\} \quad \{1,2,3,4\} \quad \{3,4\}$$

This experiment is conducted in a grid search loop in the case of polynomial and radial basis function kernels to find the best parameters of the SVM in terms of kernel scale and box constraint. The box constraint is a parameter that controls the weighting of the samples that are outside the SVM margins when performing the training to minimize the cost of the samples.

The best-performing configurations identified for the individual primitives were chosen for our system. Details about the configurations and the classifiers used are listed in Table 5.1. The used configurations lead to around 79% accuracy for planes, 97% for spheres and 89% for cylinders. The number of support vectors give an indication about the complexity of the problem to be solved by the SVM. While the number of support vectors is very low for spheres, it is considerably higher for cylinders.

The planes are the hardest to separate, as about 16% of the samples are supporting the class separation surface. A reason for this is that there is a greater variety of planes than the other two shapes, making then more challenging to separate. The set of chosen features is still not fully suitable to describe planes and capture their inherent properties.

Classifier	Kernel	Feature set	# Training Samples	# Support Vectors	Accuracy
Plane	rbf	{1, 3, 4, 5}	3008	507	79.85%
Sphere	polynomial	{1, 2, 3, 4}	1491	30	97.57%
Cylinder	rbf	{1, 2, 3, 4, 5}	1582	258	89.3%

Table 5.1: SVM classifiers properties and accuracy.

6 Experiments

In this chapter, we present experimental results. We describe the experimental setup for data acquisition, as well as the generation and properties of artificial data. The performance of the algorithm is measured using precision and recall when detecting primitives in a scene. Moreover, we show video frames with rendered detected primitives, like they would be used in an AR application.

6.1 Datasets

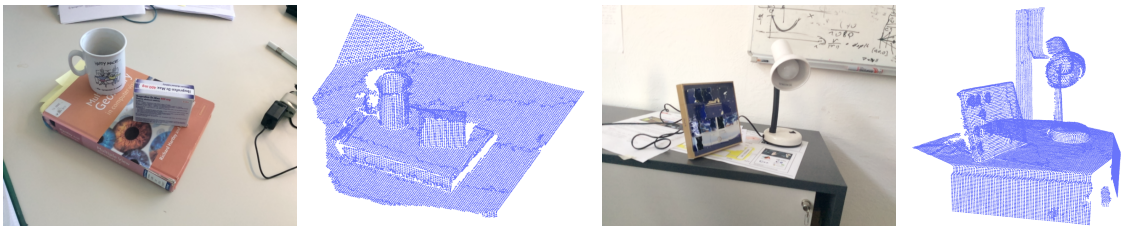
6.1.1 Real World Datasets

Figure 6.1: Ipad with attached Structure sensor.

We recorded static indoor scenes with the help of an Ipad with an attached Structure sensor [45]. The sensor uses infrared structured light to recover depth by projecting a non-uniform pattern of dots in the scene [46]. It calculates a 640 × 480 pixels depth map that is then back projected to 3D obtaining a point cloud. We work directly on point clouds,

already registered in a global coordinate system. The registration is computed locally on the Structure sensor, by a recording application using the Structure sensor Software Development Kit (SDK).

The acquired data sets encompass between 50 and 300 frames. Per frame, there a unstructured point cloud containing between 16,000 and 30,000 points registered in the global coordinate system, the camera pose and an image of the scene. The intrinsic parameters of the color camera are also available. The span of the cloud is of approximately 1 cubic meter because of the size of the scan-cube of the Structure sensor software. However, the algorithm is not dependent on this restriction.



(a) **(b)** **(c)** **(d)**

Figure 6.2: Samples of scenes captured with the Structure sensor, showing images of the scenes with corresponding point clouds.

Samples of captured data can be seen in Figure 6.2. The point cloud at a given frame is obtained from merging and downsampling all the previous point clouds up to the current frame. It contains noise that decreases in time, as more points are accumulated because of the camera movement. The internal implementation of the data acquisition in the Structure sensor is gradually increasing the accuracy of the reconstruction. The noise also increases with depth. A pixel in the depth map offers less precise depth information the further away the objects are from the sensor. More detailed properties of the Structure sensor are discussed later in the Results Section 6.2.

We aim to capture various scene setups with diverse geometric properties, needed to extensively test the designed algorithm, focusing on man-made objects, as the primitive fitting approach is suitable for such scenarios.

6.1.2 Synthetic Datasets

6.1.2.1 Generation

In order to evaluate the approach on data with no noise for a baseline performance, clean datasets are needed, containing primitives with known parameters. Therefore we generate artificial datasets, according to the pipeline seen in Figure 6.3.

First, a scene is created in Blender manually, and a camera path is specified. Then, a script is run that outputs the rendered images from Blender for all frames, along with the Z-buffer output. The depth map is saved in OpenEXR format, so that raw depth values

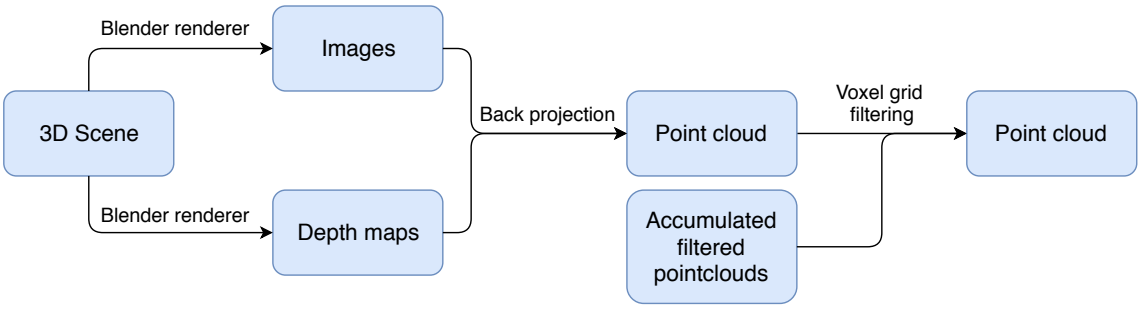


Figure 6.3: Steps required for generating artificial datasets.

are preserved. An example is shown in Figure 6.4a. Along with the frames, the camera parameters are also saved.

For each frame, a 3D reconstruction of the scene is performed, as described by Equations 6.1, 6.2 and 6.3. Given the camera intrinsics K, the extrinsics as rotation matrix R and camera center C, and the depth per pixel as distance from the camera plane to world point, we back project the points x from the camera plane back to the real world point, X_{world}. From the depth between the world point and the camera plane, we calculate the depth from the world point to the camera center with the use of triangle relations. Afterwards, we back project the points to 3D world coordinates, obtaining a point cloud as seen in 6.4b. The back projection is computed as follows:

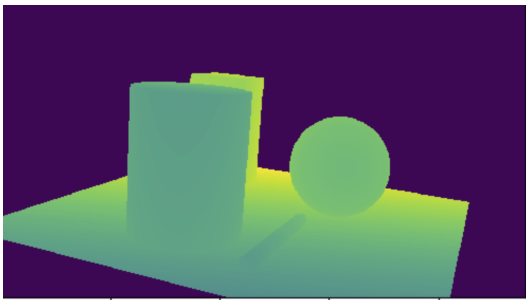
$$\mathbf{ray} = K^{-1} \cdot x \tag{6.1}$$

$$X_{camera} = \mathbf{ray} \cdot depth \tag{6.2}$$

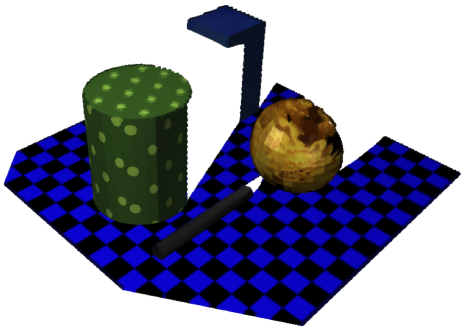
$$X_{world} = (R' \cdot X_{camera}) + C \tag{6.3}$$

We sample the points at high resolution, in comparison to what is needed. A downsampling is needed as the density of the points is higher in the regions closer to the camera, because there are more pixels to back project per unit of area. To solve this, we apply a modified voxel grid filter on each frame. In the voxel grid filter, a grid of voxels is overlapped on the scene, and the mean point position is taken as an output for each voxel. As we do not want to introduce any noise into the point cloud during downsampling, it is not the mean point position per voxel that is saved, but the position of the point closest to the center of gravity of the voxel. An output of this step is shown in Figure 6.4c. This step does not only downsample the cloud, but also ensures a more even distribution on the point on the surface, as pointed out by Hackel et al. [22], where a voxel grid filter is also used for pre-processing a point cloud before semantic segmentation.

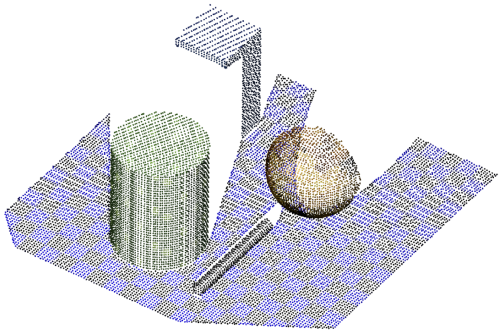
Because point cloud properties that resemble a real-world dataset are preferred, the clouds have to be merged together over the frames. The structure of the scene needs to accumulate points from a multitude of viewpoints, without changing the density of the point clouds. For example, if one would turn the camera around a cylinder in a circular



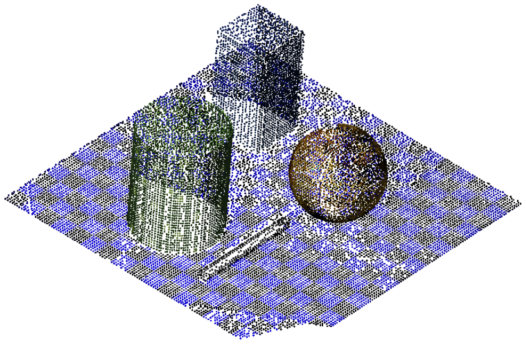
(a) Depth map in OpenEXR format representing distances from objects to the camera plane.



(b) Reconstructed point cloud.



(c) Downsampled point cloud with the VoxelGrid filter.



(d) Point cloud obtained from merging point clouds from previous frames together with the current frame.

Figure 6.4: Depth map, color image and point cloud of a frame of an artificial dataset.

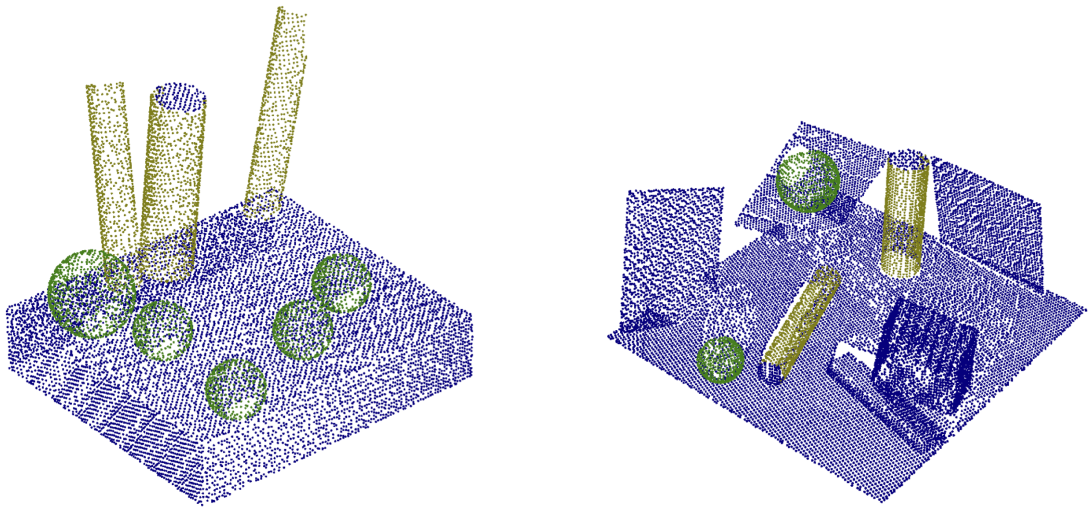
trajectory, in the end one would have a point cloud of the whole cylinder, and not just a part of it. We merge point clouds obtained at previous frames and downsample the results according to Formula 6.4, where P^k is a point cloud corresponding to frame k, and '+' denotes point cloud merging:

$$P^k_{filtered} = VoxelGridFilter(P^k + P^{k-1}_{filtered}) \tag{6.4}$$

To increase the density of the points along the frames, an extra voxel grid filter with increasingly large voxel sizes is applied on the whole dataset. Results of this step can be seen in Figures 6.4c and 6.4d.

The artificial datasets encompass sets of 99 frames, with 9000 to 30,000 points per frame. Similarly to the real-world datasets, per frame we generate a point cloud, that

is registered in a global coordinate system, a camera pose and an image of the scene. Intrinsic parameters of the virtual camera are provided by Blender.



(a) **(b)**

Figure 6.5: Examples of labeled artificial point clouds.

6.1.2.2 Limitations

The synthetic datasets are used as ground truth; however, they have some defects. The density of the points is not the same on all surfaces, because of the 3D reconstruction and of the scene discretization in pixels in 2D. In Blender, the primitives are meshes, which means that they are represented by faces, so the sphere and cylinder surface are not perfectly smooth. Therefore, the sampled points on the shape faces are not satisfying the parametric equations exactly, but are approximations.

6.1.2.3 Labeled Artificial Datasets

The 3D point clouds used later for training and testing need to be labeled, so that a point carries the label of the type of primitive it comes from. Because of the labels, we create primitives in Blender that are color-coded: planes are blue, sphere are green and cylinders are yellow. When performing the 3D reconstruction the color of the primitives is transferred to the points, representing the primitive label. The cylinder caps are labeled as planes, as seen in Figure 6.5. The expected behavior of our approach is to fit planes to the cylinder caps; the points on the caps are not taken into account in the minimization problem when fitting a cylinder.

6.2 Results

We evaluate the performance of the algorithm in terms of precision and recall, on both real and artificial datasets.

The precision and recall achieved when running the algorithm are calculated as follows for a dataset: The algorithm is run on the whole dataset, resulting in sets of fit primitives $P_{detected}$ for each frame. The set of fit primitives for the last frame is manually inspected and adjusted to correspond to the expected ground-truth primitives. A reference primitive set $\mathrm{P}_{\mathrm{ref}} = p_1^{\mathrm{ref}}, p_2^{\mathrm{ref}}, ..., p_n^{\mathrm{ref}}$ is obtained.

For each frame, the set of detected primitives $P_{detected}$ is compared to the reference primitives P_{ref}. The comparison is done by using the merging criteria described in Section 4.5. The results encompass correctly detected planes $\mathcal{P}_{planes}$, correctly detected spheres $\mathcal{P}_{spheres}$ and correctly detected cylinders $\mathcal{P}_{cylinders}$.

Precision and recall are calculated as follows:

$$\text{Precision} = \frac{|\mathcal{P}_{planes}| + |\mathcal{P}_{spheres}| + |\mathcal{P}_{cylinders}|}{|P_{detected}|} \tag{6.5}$$

$$\text{Recall} = \frac{|\mathcal{P}_{planes}| + |\mathcal{P}_{spheres}| + |\mathcal{P}_{cylinders}|}{|P_{ref}|} \tag{6.6}$$

The precision and recall per shape class is calculated using the same principle, namely dividing the number of true positives by the number of true positives and false positives for precision and diving the number of true positives by the number of relevant samples for recall.

6.2.1 Real Datasets

Using the trained SVMs, we tested our algorithm on a total of eight scenes with known object parameters. The performances obtained are listed in Table 6.1.

Planes reach the poorest precision due to the fact that they are chosen to approximate any shape. Cylinders achieve a higher precision, but still lower than the spheres. The reason is likely that cylinders can reasonably well approximate both spheres and parts of planes, leading to ambiguities in certain regions. Spheres reach a high precision, due to a sphere's neighborhood more specific curvature, which makes it harder to mistake a sphere for a cylinder and even harder to mistake it for a plane.

In terms of recall, planes perform the best, while cylinders and spheres reach similar results. This means that the planes are the easiest to identify.

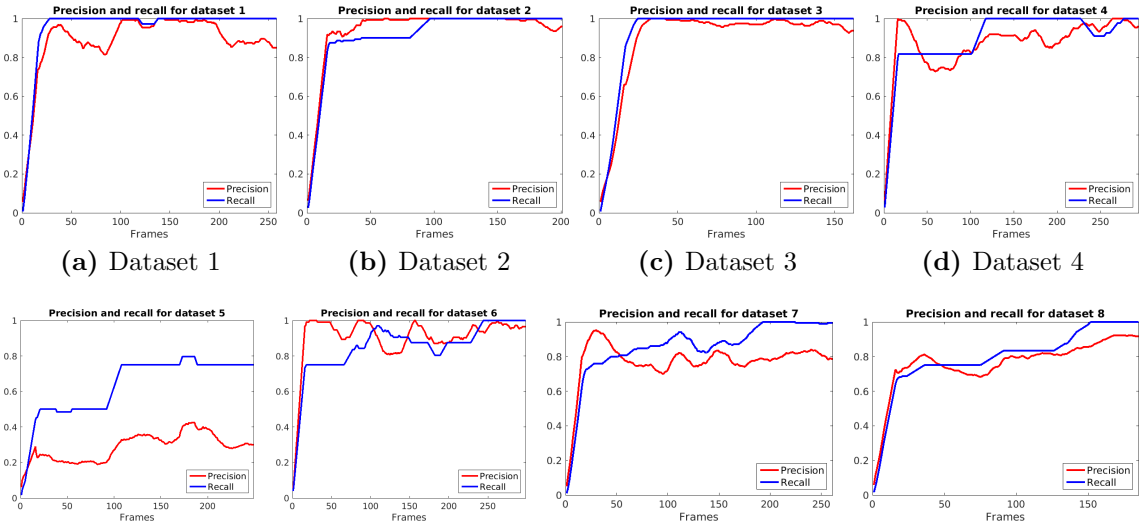
In Figure 6.6, the average-filtered precision and recall achieved when running the algorithm on each individual dataset is depicted. One can see that both precision and recall are low in the beginning and steeply increase over the first few frames, as the point clouds get denser, and new geometry is revealed. Precision tends to stay the same or to decrease slightly. This is due to the fact that the algorithm tries to fit shapes to remaining parts of

Dataset	Primitive	Precision	Recall
Dataset 1	Planes	0.874	0.997
	Cylinders	0.982	0.976
	Total	0.914	0.988
Dataset 2	Planes	0.976	0.940
	Spheres	1	1
	Cylinders	0.985	0.975
	Total	0.980	0.953
Dataset 3	Planes	0.928	0.995
	Cylinders	0.969	0.963
	Total	0.945	0.981
Dataset 4	Planes	0.910	0.919
	Spheres	1	1
	Total	0.899	0.926
Dataset 5	Planes	0.224	1.000
	Spheres	0.959	0.321
	Cylinders	0.293	0.980
	Total	0.295	0.656
Dataset 6	Planes	0.914	0.841
	Spheres	1	0.993
	Cylinders	0.997	1
	Total	0.934	0.880
Dataset 7	Planes	0.797	0.878
	Cylinders	0.867	0.977
	Total	0.796	0.885
Dataset 8	Planes	0.772	0.814
	Spheres	1	0.989
	Cylinders	0.971	0.908
	Total	0.801	0.836
Total	Planes	0.796	0.921
	Spheres	0.994	0.859
	Cylinders	0.821	0.885

Dataset	Primitive	Precision	Recall
Artificial dataset 1	Planes	0.926	0.931
	Spheres	1	1
	Cylinders	0.943	0.944
	Total	0.936	0.942
Artificial dataset 2	Planes	0.920	0.987
	Spheres	1	0.975
	Cylinders	0.973	1
	Total	0.937	0.987
Artificial dataset 3	Planes	0.936	0.843
	Spheres	0.997	0.960
	Cylinders	0.591	0.995
	Total	0.858	0.873
Artificial dataset 4	Planes	0.979	0.904
	Spheres	1	1
	Cylinders	0.968	0.995
	Total	0.981	0.947
Artificial dataset 5	Planes	0.972	0.997
	Cylinders	0.918	0.960
	Total	0.940	0.979
Artificial dataset 6	Spheres	1	1
	Total	0.998	1
Artificial dataset 7	Planes	1	0.870
	Total	0.997	0.870
Artificial dataset 8	Planes	1	0.834
	Spheres	1	1.000
	Cylinders	1	1
	Total	1	0.859
Total	Planes	0.960	0.909
	Spheres	0.999	0.989
	Cylinders	0.894	0.982

Table 6.1: Results of experiments in terms of precision and recall, on real (left) and artificial (right) datasets.

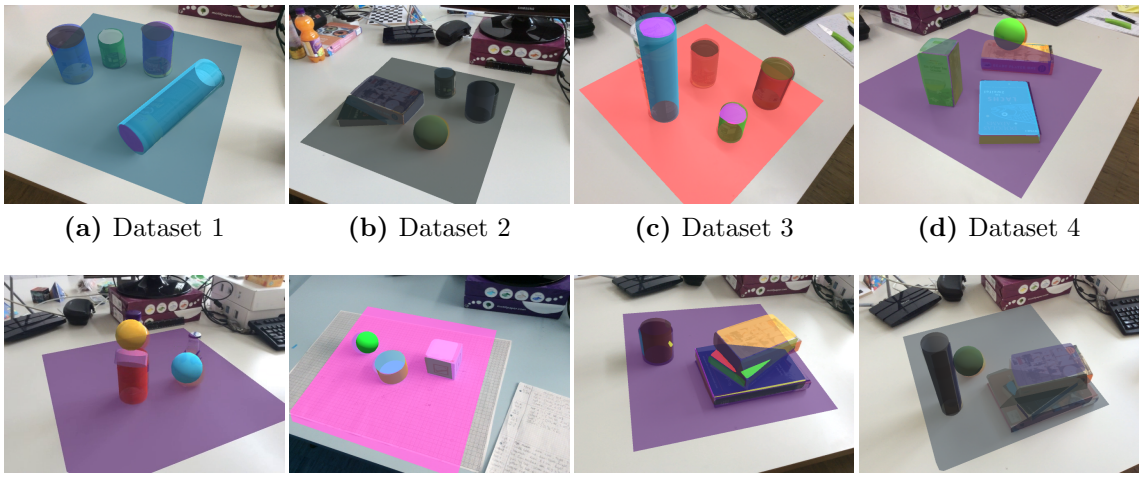
the point cloud that are not captured completely by the sensor yet, or that cannot be fully modeled by our approach. This is discussed in more detail in Section 7.1.5. The algorithm tries to fit different primitives to newly discovered or freed areas in the point cloud, while some of these fit primitives are not correct. Recall increases as expected, reflecting the



(a) Dataset 1 **(b)** Dataset 2 **(c)** Dataset 3 **(d)** Dataset 4

(e) Dataset 5 **(f)** Dataset 6 **(g)** Dataset 7 **(h)** Dataset 8

Figure 6.6: Average-filtered precision and recall for real datasets 1-8.



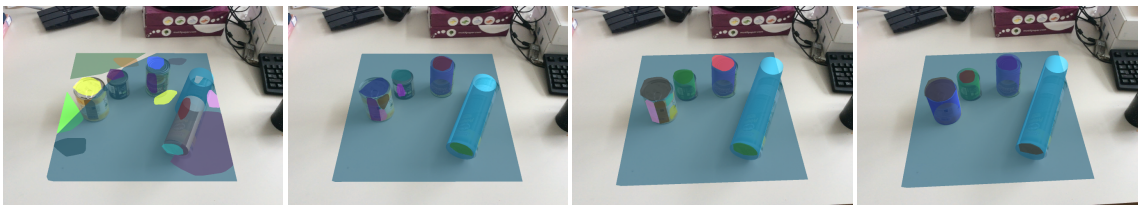
(a) Dataset 1 **(b)** Dataset 2 **(c)** Dataset 3 **(d)** Dataset 4

(e) Dataset 5 **(f)** Dataset 6 **(g)** Dataset 7 **(h)** Dataset 8

Figure 6.7: Results showing overlaid detected primitives in selected frames of real datasets 1-8.

successful retrieval of the primitives over time. This is an expected behavior because, over time, the point cloud expands, providing more geometrical details, and therefore the fit also improves.

Results of the algorithm are shown in Figures 6.7 and 6.8. The latter emphasizes the improvement of the fit over time.

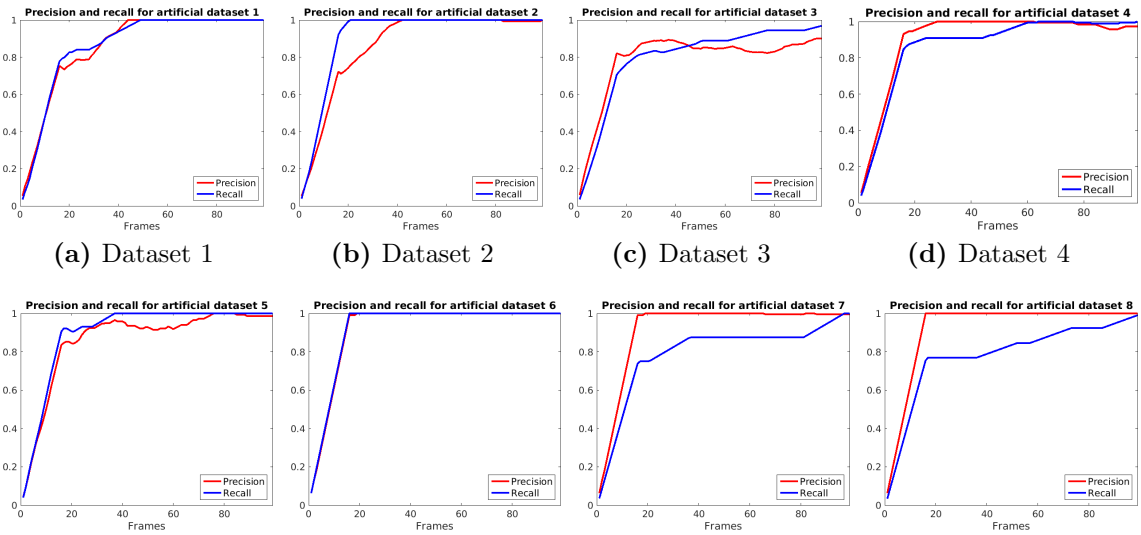


(a) Frame 1 **(b)** Frame 4 **(c)** Frame 10 **(d)** Frame 14

Figure 6.8: Results showing the improvement of primitive fitting over time in selected frames of real dataset 1.

6.2.2 Synthetic Datasets

For comparison, the algorithm is also run on artificially created datasets of scenarios similar to the real datasets. The thresholds are empirically tuned for the artificial datasets, according to scale and to the noise level. For the artificial data, the noise level is expected to be very low.



(a) Dataset 1 **(b)** Dataset 2 **(c)** Dataset 3 **(d)** Dataset 4

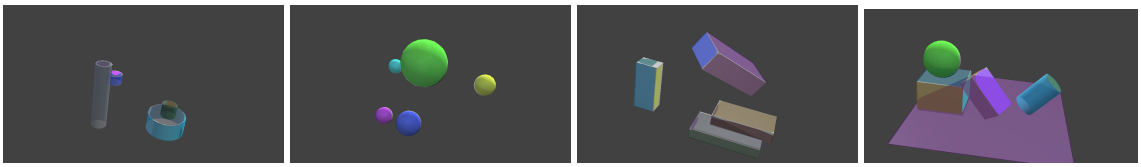
(e) Dataset 5 **(f)** Dataset 6 **(g)** Dataset 7 **(h)** Dataset 8

Figure 6.9: Average-filtered precision and recall for artificial datasets 1-8.

Looking at the precision and recall curves as shown in Figure 6.9, they ascend more smoothly and with less oscillations than in the real-world datasets. This is the expected behavior of the algorithm when running on artificial data, due to the nature of the objects and to the low noise level.

Here cylinders reach the smallest precision, which means that they are overdetected. This effect was also partially reflected in the results of the real-world dataset. In contrast to the real datasets, however, planes achieve the lowest recall, but the recall for all shapes

(a) Dataset 1 **(b)** Dataset 2 **(c)** Dataset 3 **(d)** Dataset 4



(e) Dataset 5 **(f)** Dataset 6 **(g)** Dataset 7 **(h)** Dataset 8

Figure 6.10: Results showing overlaid detected primitives in selected frames of artificial datasets 1-8.

is over 0.9, which indicates the success of the method. Selected frames from the artificial datasets with overlaid detected primitives are shown in Figure 6.10.

6.3 Experimental Setup

All experiments are run in a desktop setup, on a Lenovo P50 laptop with an eight core Intel Core i7 CPU running at 2.60 GHz. The experiments are run solely on the CPU, partly parallelized.

Most of the software is written as MATLAB scripts, using MATLAB 2015b. The artificial data generation and the processing of the datasets are using Python 3.5 and the Point Cloud Library [47] Python wrapper [48].

The non-linear optimization using the Ceres solver is written in C++. This is compiled into MEX files, then called from MATLAB.

7 Discussion

In this chapter, we discuss challenges faced by the algorithm, both from a hardware and software point of view. We show concrete scenarios where known weaknesses are displayed as well as present ideas for further improvement of the method.

7.1 Known Limitations

7.1.1 Sensor Flaws

As any structured light sensor, the Structure sensor has difficulties dealing with transparent objects; namely, they are not captured at all. An example is shown in Figure 7.1. Also highly concave regions of point clouds are inaccurately captured due to reflections [49]. An example of noise introduced by reflections is shown in Figure 7.2, where the bottom part of the sphere is reconstructed at a wrong depth.

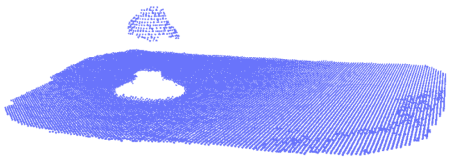
From our experiments, we deduced that the Structure sensor captures noisy points on red object surfaces. The source code that calculates depth vales inside of the sensor is not publicly available; we assume that the quality of points on the surface of red objects is poorer because of the use of infrared light. With red objects, it is harder to distinguish infrared light from the red object surface color.

It is a known weakness of structured light sensors that they are sensitive to surface reflectance and to the scene color, as well as scene illumination [49].

7.1.2 Normals Orientation Ambiguity

The Structure sensor captures point clouds and merges them locally in each frame. As the surface normals are calculated after this step from combined point clouds merged from more views and as there is no correspondence between previous point clouds, we cannot know where the sensor was located when capturing a certain point p_i. Therefore, the direction of the normals does not provide accurate information about the inside and outside of objects. This is a possible source of error when performing segmentation based

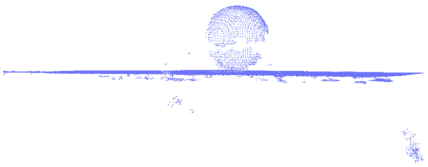
(a) Scene containing a transparent object.



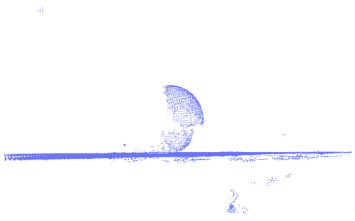
(b) The point cloud of the scene captured with the Structure sensor.

Figure 7.1: The structured light sensor cannot capture transparent objects; the bottle (left) is missing from the point cloud, and its only non-transparent part, the sticker, was captured (right).

(a) Scene containing a reflective sphere.



(b) Front view of captured point cloud.



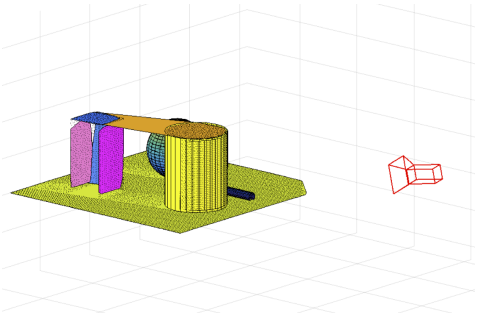
(c) Side view of captured point cloud.

Figure 7.2: Scene containing a highly reflective sphere that causes noise in the point cloud.

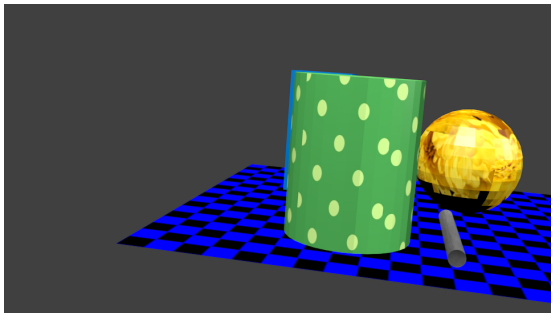
on normal orientation. A solution would imply keeping track of the sensor position when capturing each point in the point cloud.

7.1.3 Merging Due to Collinearity between Camera and Primitives

A faulty merging happens when primitives with similar parameters are aligned with the camera, as shown in Figure 7.3, while performing a label propagation and search radius expansion step. If the camera and the primitives are collinear, the inliers of the primitives are projected onto the camera plane very closely to each other. As the 2D search region of primitive s_k is increased, it overlaps the inliers of primitive s_l. Because the primitives of s_l pass the inlier criterion using the parameters of s_k, they are wrongfully assigned to s_k. This newly expanded primitive s_k is sometimes rejected by the surface inliers distribution criterion described in Section 4.2.5.3. However, because the threshold is permissive, also primitives with multiple surface inliers clusters are accepted, like the orange plane in Figure 7.3a.

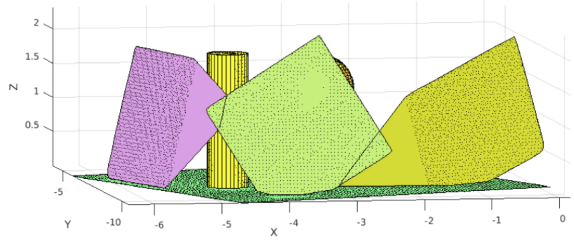


(a) Scenario when two horizontal planes and the camera are collinear.

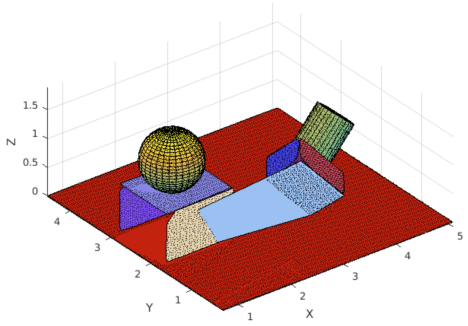


(b) Color image of the scene from the perspective of the camera.

Figure 7.3: Erroneous plane merging example in artificial dataset 4, because of the proximity of the two planes in the image that leads to a wrong label propagation.



(a) Faulty plane expansion, as inliers of the middle plane are wrongfully assigned to the side planes.



(b) Faulty plane expansion, as the inliers of the vertical pink plane pass the inliers criterion of the light blue plane.

Figure 7.4: Primitive search radius expansion leading to wrongfully assigned inliers for planes in artificial datasets 3 and 8.

The inlier expansion can also fail in case a primitive expands over another primitive, as there is no check whether the new inliers are spatially close enough to the cluster formed by the inliers of a primitive. This is illustrated in Figure 7.4. Calculating the distance to the nearest inlier neighbors when adding an inlier to a primitive would solve this problem, but add computational complexity to the algorithm.

7.1.4 Convex Hulls Implicit Restrictions

All planar surfaces are approximated by their convex hulls, obtaining *plane segments*. This is a potential cause of inaccurate results.

Such a situation can occur when merging planes, because it is possible that planar surfaces are not supported by convex objects. In such a case, the object surfaces would be concave, but they are merged, because their convex hulls are very close to each other

or overlap each other in regions where there are no inliers.

Because of this effect, one of the future improvements of the work includes handling concave polygons when dealing with plane segments, which would make the approach more exact.

7.1.5 Non-expressible Objects

The more regular objects are, the easier it is to represent them as geometric primitives. When the geometry is complex or ambiguous, the proposed algorithm tries to model it with the known primitives, becoming locally unstable in-between frames by oscillating between chosen primitive types.

This behavior is shown in Figure 7.5, running on the real-world dataset 5. The lowest precision and recall performance is achieved from all tested datasets, because the shapes of the two bottles are ambiguous and challenging to model with the primitives. Even for a human, it is hard to tell whether the bottles are, in reality, spheres or cylinders.

(a) Frame 70 **(b)** Frame 91 **(c)** Frame 93 **(d)** Frame 104

Figure 7.5: Selected frames from real dataset 5, showing the unstable behavior of the algorithm when dealing with objects having challenging shapes.

7.1.6 Contextual Relations

In real-world datasets 2, 7 and 8, the edges of different stacked books are partly merged together into the same plane. This is because the algorithm is relying on the geometry only, without any further context regarding object separation. Thus the algorithm is unable to separate object instances, as shown in Figure 7.6. A possible improvement is the addition of contextual relations or specific constraints between primitives [16, 33].

7.1.7 Small Objects Detection and Classification

The current implementation faces challenges when modeling small objects. This is mainly due to the segmentation step based on the orientation of surface normals. In case of modeling small spheres and cylinders, even when they have a locally smooth surface, the angles between their normals are above the acceptable thresholds for performing segmentation reliably. This leads to categorizing the inliers as cluster edges instead of surfaces. Consequently, those points are not taken into consideration for primitive fitting. A possible

(a) (b)

Figure 7.6: Ambiguous plane merging on the side of the stacked books in real datasets 7 and 8.

improvement is the use of adaptive thresholds for the geometric segmentation, dependent on cluster scale and expected noise level. Adaptive thresholds will likely improve the overall performance of the algorithm, as the segmentation step has crucial influences on the detection of primitives.

Another issue with small objects is that they are rejected more often by the classifiers because small primitives are often fit wrongly in an attempt to approximate noisy regions. Therefore, they are also learned by the classifiers as wrong fits. A solution would be the introduction of classes based on size, i.e. small cylinder and large cylinders [34].

7.2 Possible Improvements

7.2.1 Runtime

Although our current implementation is running close to real-time on desktop hardware, it is not ready for application at interactive rates on mobile devices yet.

Efficiency improvements such as an efficient calculation of surface normals from depth maps [3] would improve the runtime. The algorithm also has high parallelization potential. The primitive fitting, refinement, and classification steps can be parallelized due to the pre-segmentation, because segmented point cloud regions are processed independently.

Similar approaches that run in real time [17, 18] provide a hint that our method can achieve this. The only possible bottleneck, which is the SVM classification, is a fast procedure and would not represent a problem.

7.2.2 Machine Learning Models

Inspired by recent work progress in ML presented in Section 2.2 and 2.3 and by our own experiments with the SVM, we think that using a different approach for classification such as CNNs or CRFs would be useful for separating classes more effectively. Moreover, we

think that experimenting with more features like the geometric neighborhood features used in [22] or FPFH could also improve the classification results.

7.3 Conclusion

We introduce an approach for structural modeling of indoor static scenes using planes, spheres and cylinder as geometrical primitives. The method is capable of inferring the missing geometry from point clouds in an incremental way. Because the user moves the sensor to reveal new geometrical properties of the captured objects, the algorithm is can discard outdated fits automatically based on outputs of SVM classifiers, improving the modeled structure of the scene over time.

Being suited for a stream of point clouds incoming from a mobile device, the system outputs a compact representation of the scene as a set of parametric models for each frame. This primitive set is ready to be further used by AR applications that, considering the recent progress in the field, will surely be a part of our near future.

A

List of Acronyms

AR	Augmented Reality
BIM	Building Information Modelling
CAD	Computer Aided Design
CNN	Convolutional Neural Network
CRF	Conditional Random Field
DBSCAN	Density Based Spatial Clustering of Applications with Noise
FPFH	Fast Point Feature Histogram
ICP	Iterative Closest Point
ML	Machine Learning
MLE	Maximum Likelihood Estimation
MLESAC	Maximum Likelihood Consensus
MSAC	M-estimator Sampling and Consensus
RANSAC	RANdom SAmpling Consensus
SDK	Software Development Kit
SLAM	Simultaneous Localization and Mapping
SVM	Support Vector Machine

Bibliography

[1] Andrew T Miller and Peter K Allen. Graspit! a versatile simulator for robotic grasping. *IEEE Robotics & Automation Magazine*, 11(4):110–122, 2004. (page 2, 34)

[2] Radu Bogdan Rusu, Nico Blodow, Zoltan Csaba Marton, and Michael Beetz. Close-range scene segmentation and reconstruction of 3d point cloud maps for mobile manipulation in domestic environments. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 1–6. IEEE, 2009. (page 2, 5, 6, 14, 15, 19, 31)

[3] Dirk Holz, Stefan Holzer, Radu Bogdan Rusu, and Sven Behnke. Real-time plane segmentation using rgb-d cameras. In *Robot Soccer World Cup*, pages 306–317. Springer, 2011. (page 2, 6, 11, 15, 19, 33, 55)

[4] Andre Ückermann, Christof Elbrechter, Robert Haschke, and Helge Ritter. 3d scene segmentation for autonomous robot grasping. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 1734–1740. IEEE, 2012. (page 2, 7, 15, 19, 20, 27)

[5] F. Remondino, D. Lo Buglio, N. Nony, and L. De Luca. Detailed Primitive-Based 3d Modeling of Architectural Elements. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 285–290, July 2012. (page 2)

[6] Sven Oesau, Florent Lafarge, and Pierre Alliez. Indoor scene reconstruction using feature sensitive primitive extraction and graph-cut. *ISPRS Journal of Photogrammetry and Remote Sensing*, 90:68–82, 2014. (page 8, 13, 15)

[7] Jianxiong Xiao and Yasutaka Furukawa. Reconstructing the world's museums. *International journal of computer vision*, 110(3):243–258, 2014. (page 8, 14, 15)

[8] Sebastian Ochmann, Richard Vock, Raoul Wessel, and Reinhard Klein. Automatic reconstruction of parametric building models from indoor point clouds. *Computers & Graphics*, 54:94–103, 2016. (page 2, 13, 14, 15)

[9] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient ransac for point-cloud shape detection. In *Computer graphics forum*, volume 26, pages 214–226. Wiley Online Library, 2007. (page 2, 5, 6, 7, 8, 13, 14, 15, 27)

[10] Jonathan Ventura and Tobias Hollerer. Online environment model estimation for augmented reality. In *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*, pages 103–106. IEEE, 2009. (page 2)

[11] Renato F Salas-Moreno, Ben Glocken, Paul HJ Kelly, and Andrew J Davison. Dense planar slam. In *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*, pages 157–164. IEEE, 2014. (page)

[12] Anuruddha Hettiarachchi and Daniel Wigdor. Annexing reality: Enabling opportunistic use of everyday objects as tangible proxies in augmented reality. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 1957–1967. ACM, 2016. (page 2, 7, 15)

[13] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In *Readings in computer vision*, pages 726–740. Elsevier, 1987. (page 2, 5, 23)

[14] Paul VC Hough. Method and means for recognizing complex patterns, December 18 1962. US Patent 3,069,654. (page 2, 8)

[15] E Grilli, F Menna, and F Remondino. A review of point clouds segmentation and classification algorithms. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci*, 42(2):W3, 2017. (page 2, 14)

[16] Yangyan Li, Xiaokun Wu, Yiorgos Chrysathou, Andrei Sharf, Daniel Cohen-Or, and Niloy J Mitra. Globfit: Consistently fitting primitives by discovering global relations. In *ACM Transactions on Graphics (TOG)*, volume 30, page 52. ACM, 2011. (page 6, 15, 33, 54)

[17] Keisuke Tateno, Federico Tombari, and Nassir Navab. Real-time and scalable incremental segmentation on dense slam. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 4465–4472. IEEE, 2015. (page 7, 15, 19, 21, 30, 55)

[18] Rafael Roberto, Hideaki Uchiyama, João Paulo Lima, Hajime Nagahara, Rinichiro Taniguchi, and Veronica Teichrieb. Incremental structural modeling on sparse visual slam. *IPSJ Transactions on Computer Vision and Applications*, 9(1):5, 2017. (page 8, 14, 15, 55)

[19] Rafael Roberto, João Paulo Lima, Hideaki Uchiyama, Clemens Arth, Veronica Teichrieb, Rinichiro Taniguchi, and Dieter Schmalstieg. Incremental structural modeling based on geometric and statistical analyses. In *IEEE Winter Conf. on Applications of Computer Vision (WACV)*, pages 1–8, 2018. (page 8, 15, 27)

[20] Kristiyan Georgiev, Motaz Al-Hami, and Rolf Lakaemper. Real-time 3d scene description using spheres, cones and cylinders. *arXiv preprint arXiv:1603.03856*, 2016. (page 9, 15)

[21] Liangliang Nan, Ke Xie, and Andrei Sharf. A search-classify approach for cluttered indoor scene understanding. *ACM Transactions on Graphics (TOG)*, 31(6):137, 2012. (page 9, 15, 19)

[22] Timo Hackel, Jan D Wegner, and Konrad Schindler. Fast semantic segmentation of 3d point clouds with strongly varying density. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Prague, Czech Republic*, 3:177–184, 2016. (page 10, 15, 43, 56)

[23] Federico Tombari, Samuele Salti, and Luigi Di Stefano. Unique signatures of histograms for local surface description. In *European conference on computer vision*, pages 356–369. Springer, 2010. (page 10)

[24] Andrea Frome, Daniel Huber, Ravi Kolluri, Thomas Bülow, and Jitendra Malik. Recognizing objects in range data using regional point descriptors. *Computer vision-ECCV 2004*, pages 224–237, 2004. (page 10)

[25] Lyne P Tchapmi, Christopher B Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. Segcloud: Semantic segmentation of 3d point clouds. *arXiv preprint arXiv:1710.07563*, 2017. (page 10, 15, 37)

[26] Timo Hackel, Nikolay Savinov, Lubor Ladicky, Jan D Wegner, Konrad Schindler, and Marc Pollefeys. Semantic3d. net: A new large-scale point cloud classification benchmark. *arXiv preprint arXiv:1704.03847*, 2017. (page 10)

[27] Iro Armeni, Ozan Sener, Amir R Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1534–1543, 2016. (page 10, 13, 15)

[28] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *European Conference on Computer Vision*, pages 746–760. Springer, 2012. (page 10)

[29] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013. (page 10)

[30] John McCormac, Ankur Handa, Andrew Davison, and Stefan Leutenegger. Semanticfusion: Dense 3d semantic mapping with convolutional neural networks. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 4628–4635. IEEE, 2017. (page 10, 37)

[31] Anil Armagan, Martin Hirzer, and Vincent Lepetit. Semantic segmentation for 3d localization in urban environments. In *Urban Remote Sensing Event (JURSE), 2017 Joint*, pages 1–4. IEEE, 2017. (page 10)

[32] Radu Bogdan Rusu, Andreas Holzbach, Nico Blodow, and Michael Beetz. Fast geometric point labeling using conditional random fields. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 7–12. IEEE, 2009. (page 11, 14, 15)

[33] Thanh Nguyen, Gerhard Reitmayr, and Dieter Schmalstieg. Structural modeling from depth images. *IEEE transactions on visualization and computer graphics*, 21(11):1230–1240, 2015. (page 11, 13, 14, 15, 31, 54)

[34] Jing Huang and Suya You. Detecting objects in scene point cloud: A combinational approach. In *3DTV-Conference, 2013 International Conference on*, pages 175–182. IEEE, 2013. (page 11, 12, 14, 15, 55)

[35] Xuehan Xiong and Daniel Huber. Using context to create semantic 3d models of indoor environments. In *BMVC*, pages 1–11, 2010. (page 11, 14, 15, 19, 33, 39)

[36] Xuehan Xiong, Antonio Adan, Burcu Akinci, and Daniel Huber. Automatic creation of semantically rich 3d building models from laser scanner data. *Automation in Construction*, 31:325–337, 2013. (page 12, 15)

[37] Hema S Koppula, Abhishek Anand, Thorsten Joachims, and Ashutosh Saxena. Semantic labeling of 3d point clouds for indoor scenes. In *Advances in neural information processing systems*, pages 244–252, 2011. (page 12, 14, 15, 19)

[38] Jing Huang and Suya You. Point cloud matching based on 3d self-similarity. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*, pages 41–48. IEEE, 2012. (page 12)

[39] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. `http://ceres-solver.org`, 2018. (page 13, 31)

[40] Kang Chen, Yu-Kun Lai, and Shi-Min Hu. 3d indoor scene modeling from rgb-d data: a survey. *Computational Visual Media*, 1(4):267–278, 2015. (page 14)

[41] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996. (page 20)

[42] Alireza Bab-Hadiashar and David Suter. *Data segmentation and model selection for computer vision: a statistical approach*. Springer Science & Business Media, 2012. (page 22)

[43] Philip HS Torr and Andrew Zisserman. Mlesac: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78(1):138–156, 2000. (page 23, 24)

[44] Philip Schneider and David H Eberly. *Geometric tools for computer graphics*. Elsevier, 2002. (page 34)

[45] Structure sensor official website. `https://structure.io/`, 2018. (page 41)

[46] M Kalantari and M Nechifor. Accuracy and utility of the structure sensor for collecting 3d indoor information. *Geo-spatial information science*, 19(3):202–209, 2016. (page 41)

[47] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011. (page 50)

[48] Python wrapper for the point cloud library source. `https://github.com/strawlab/python-pcl`, 2018. (page 50)

[49] Pietro Zanuttigh, Giulio Marin, Carlo Dal Mutto, Fabio Dominio, Ludovico Minto, and Guido Maria Cortelazzo. *Time-of-flight and structured light depth cameras*. Springer, 2016. (page 51)