MapSense: Grammar-Supported Inference of Indoor Objects from Crowd-Sourced 3D Point Clouds

MOHAMED ABDELAAL, SURIYA SEKAR, FRANK DÜRR, KURT ROTHERMEL, Institute of Parallel and Distributed Systems, University of Stuttgart, Stuttgart, Germany
SUSANNE BECKER, DIETER FRITSCH, Institute for Photogrammetry, University of Stuttgart, Stuttgart, Germany

Recently, indoor modeling has gained increased attention thanks to the immense need for realizing efficient indoor location-based services. Indoor environments de facto differ from outdoor spaces in two aspects: spaces are smaller and there are many structural objects such as walls, doors, and furniture. To model the indoor environments in a proper manner, novel data acquisition concepts and data modeling algorithms have been devised to meet the requirements of indoor spatial applications. In this realm, several research efforts have been exerted. Nevertheless, these efforts mostly suffer either from adopting impractical data acquisition methods or from being limited to 2D modeling.

To overcome these limitations, we introduce the MapSense approach that automatically derives indoor models from 3D point clouds collected by individuals using mobile devices, such as Google Tango, Apple ARKit, and Microsoft HoloLens. To this end, MapSense leverages several computer vision and machine learning algorithms for precisely inferring the structural objects. In MapSense, we mainly focus on improving the modeling accuracy through adopting formal grammars which encode design-time knowledge, i.e. structural information about the building. In addition to modeling accuracy, MapSense considers the energy overhead on the mobile devices via developing a probabilistic quality model through which the mobile devices solely upload *high-quality* point clouds to the crowd-sensing servers. To demonstrate the performance of MapSense, we implemented a crowdsensing Android App to collect 3D point clouds from two different buildings by six volunteers. The results showed that MapSense can accurately infer the various structural objects together with drastically reducing the energy overhead on the mobile devices.

CCS Concepts: • **Human-centered computing → Ubiquitous and mobile devices**.

Additional Key Words and Phrases: indoor mapping, crowd-sensing, machine learning, formal grammars, QoS-aware sensing

ACM Reference Format:
Mohamed Abdelaal, Suriya Sekar, Frank Dürr, Kurt Rothermel and Susanne Becker, Dieter Fritsch. 2019. MapSense: Grammar-Supported Inference of Indoor Objects from Crowd-Sourced 3D Point Clouds. 1, 1 (January 2019), 28 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Nowadays, there exists a growing demand for identifying indoor environments for location-awareness in large public areas, e.g. airports, hospitals, campuses, warehouses, and manufacturing

Authors' addresses: Mohamed Abdelaal, Suriya Sekar, Frank Dürr, Kurt Rothermel, first.last@ipvs.uni-stuttgart.de, Institute of Parallel and Distributed Systems, University of Stuttgart, Stuttgart, Germany; Susanne Becker, Dieter Fritsch, Institute for Photogrammetry, University of Stuttgart, Stuttgart, Germany, first.last@ifp.uni-stuttgart.de.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
XXXX-XXXX/2019/1-ART $15.00
https://doi.org/10.1145/1122445.1122456

© ACM, 2020. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in ACM Transactions on Internet of Things (TIOT), Vol. 1, No. 1, January 2020, DOI: 10.1145/3379342.

facilities. For instance in hospitals, it is necessary to localize the medical equipment, provide navigation to the visitors and monitor the exercise and location of the patients. Similarly, it is crucial for firefighter assistance applications to provide navigation services to firefighters through safe zones or even localize people in danger indoors. In such areas, adopting location-based services (LBSs) strongly requires the existence of localization technology and the generation of up-to-date indoor maps [2]. By taking into consideration the advancement of indoor localization technologies, alternative methods of generating indoor maps have to be developed. In this realm, Google launched its indoor mapping initiative through which voluntarily uploaded floor plans for solely 10,000 buildings are available on-line to its customers [19]. One major drawback of such floor plans emerges from being manually generated. Specifically, the manual generation of indoor maps is a de facto complex, time-consuming, and error-prone process involving on-site calibration, professional devices, and multi-party coordination.

With the advent of powerful mobile devices, e.g. smartphones, smartwatches, and smart glasses, the concept of crowdsensing to automatically generate indoor floor plans has been widely accepted. Mobile crowdsensing is a promising paradigm for large-scale data collection harnessing the recent advancements in ubiquitous mobile devices. Practically speaking, crowdsensing is typically performed by a group of participants, carrying smart mobile devices, who react to sensing queries from the crowdsensing server(s). In the realm of indoor modeling, several efforts have been exerted to properly exploit the embedded sensors in commercially-available mobile devices for generating precise models [11, 18, 24]. Although these aforementioned efforts predominantly achieve a reasonable modeling accuracy, they still suffer from three main shortcomings: (1) They mostly require large datasets to construct precise indoor maps. For example, Jigsaw [11] demands collecting between 100 and 150 images for each landmark to precisely model a certain area. (2) These efforts broadly overlook important quality of service (QoS) metrics, such as the energy efficiency of the participating mobile devices as well as the computational complexity. (3) The collected data types are typically limited to the derivation of 2D indoor maps.

Accordingly, there is an immense need to utilize new data types for efficient indoor modeling. In this context, 3D point clouds stand as an optimal candidate since they provide 3D representation of the environment. In a 3D point cloud, the points usually represent the X, Y, and Z geometric coordinates of an underlying sampled surface. Laser scanners have been traditionally employed to collect high-quality point clouds. Nevertheless, adopting these scanners for indoor modeling is not straightforward since they certainly increase the complexity, cost, and time of data acquisition. Hence, a challenge of efficiently collecting and processing point clouds for constructing 3D indoor models emerges.

To overcome this challenge, we introduce MapSense, a QoS-aware automatic 3D indoor modeling method using a small number of crowd-sourced 3D point clouds. MapSense leverages recently-developed mobile platforms, e.g. Google Tango, Microsoft HoloLens, and Apple ARKit, to cooperatively collect 3D point clouds (cf. Figure 1). The core idea of MapSense is to partition each point cloud into a set horizontal and vertical planes, before inferring the various indoor objects, e.g. walls, ceiling, floor, doors, and furniture. To this end, MapSense harnesses several computer vision and machine learning techniques, such as ray tracing, randomized Hough transform, and random forest classifier. However, the indoor models obtained by only considering the collected point clouds always suffer from inaccuracies. Therefore, MapSense utilizes various kinds of structural information, encoded in a *formal grammar*, to support the mapping process and, in particular, compensate for possible inaccuracies of the collected point clouds due to being captured by non-experts. Indoor grammars are powerful tools which can be used to encode structural information for different kind of architectural domains. In public buildings, a floor plan typically follows certain architectural principles. For example, in an office building, an assistant's office is very likely located next to an

executive's office and both rooms have specific dimensions. Through exploiting this knowledge, we can simply correct and complete a floor plan derived from inaccurate or incomplete data. In this article, we encode such structural information using a grammar that includes, for instance, the dimensions of rooms, the number of rooms, the relative room ordering, geometric constraints, etc.

MapSense also considers the energy efficiency of the participating mobile devices. In this realm, MapSense makes use of several energy-awareness techniques, including Octree compression, voxel grid downsampling, and probabilistic quality model. The first two techniques tackle the communication energy overhead through reducing the number of points to be reported to a crowdsensing server. Whereas, the probabilistic quality model is mainly used to prevent reporting low-quality point clouds, thus reducing the energy overhead of repeating the sensing queries. To the best of our knowledge, MapSense is the first QoS-aware mapping method which employs formal indoor grammars in combination with a probabilistic quality model to derive highly-accurate indoor models from a small set of crowd-sensed point clouds in an energy-efficient manner.

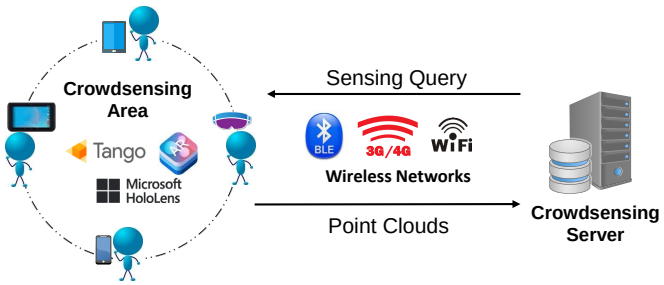


Fig. 1. Crowdsensing 3D point clouds using commercial mobile devices

In detail, we provide the following contributions: (1) We define an architectural framework and multi-step process for the automatic inference of indoor objects, e.g. walls, ceilings, floors, tables, chairs, and doors, supported by structural information. (2) We present a probabilistic quality model for assessing the quality of each captured point cloud. In this manner, we prevent uploading low-quality point clouds, thus eliminating the need for sending several identical sensing queries. To this end, we first analyze the quality factors and metrics of point clouds, before providing a formal definition of an acceptance probability through which a point clouds can be accepted/rejected. (3) We introduce a processing pipeline that, as a first step, derives an initial indoor model from a small set of inaccurate crowdsensed point clouds. In this initial step, we employ the randomized 3D Hough transform for extracting and labeling the walls, floor, and ceiling objects. Moreover, ray tracing and random forest methods are used for detecting and classifying other objects, e.g. doors, tables, and chairs. (4) We define a formal grammar through which, in a second step, the initial model is refined by compensating for inaccuracies and missing data. (5) We present a proof-of-concept implementation and evaluation in a real-world scenario. To this end, we collected more than 3 GBytes of point clouds using a set of Google Tango mobile devices. We deliberately integrated the main three features of Tango technology, namely *motion tracking*, *depth perception*, and *area learning* [5] to improve the localization accuracy while capturing the point clouds. The results show that using our grammar-based approach, we obtain a significant improvement of the modeling accuracy (at least by 76.3%) relative to the initial models. Moreover, MapSense reduces the energy overhead on the mobile devices by at least 44% compared to a "no processing" method in which the captured data is directly reported to the servers without any processing.

The rest of this article is structured as follows: Section 2 introduces the system model, the architectural framework of MapSense, and our assumptions. The various preprocessing steps

performed on the mobile devices, including the probabilistic quality model, are explained in Section 3. Section 4 explains the processing steps filtering the point clouds and then for identifying the various surface planes. Moreover, this section introduces the indoor formal grammars and how they can be efficiently used to improve the modeling accuracy of the initial models. Section 5 describes the detection of door objects using a combination of ray tracing and random forest techniques, before Section 6 presents the extraction of the surface planes corresponding to furniture, e.g. tables and chairs. In Section 7, we present our real-world evaluation and our prototype implementation that we designed for the acquisition of point clouds using mobile devices, before discussing the obtained results. Finally, Section 8 highlights the main differences between MapSense and related work, before Section 9 draws a conclusion with an outlook on future work.

2 SYSTEM OVERVIEW

In this section, we explain the various components of MapSense together with our assumptions. In general, the problem of constructing precise indoor models can be reduced to estimating three primary components, including the *geometric coordinates* of objects in the indoor environment, the *topology* relationship (i.e. connectivity) among indoor objects, and the *structural objects* which classifies and interprets the space (e.g. as a room or a corridor). Figure 2 depicts our proposed processing pipeline for estimating these components. The system comprises a set of mobile devices $\mathcal{M} = m_1, \cdots, m_k$, capable of collecting 3D point clouds. Each participating mobile device m_i typically responds to a *sensing query* through capturing a point cloud $P_{in}^{(i)}$ of the queried area. In MapSense, a sensing query q comprises the coordinates of the area to be scanned and the minimum quality requirements T_q whose value lies in the range $[0, 1]$. To be eligible for transmission, the point cloud $P_{in}^{(i)}$ has to pass a lightweight acceptance test, implemented by a probabilistic quality mode. Through this test, we make sure that $P_{in}^{(i)}$ does not suffer from intolerable inaccuracies or incompletenesses. In this manner, we eliminate the need for repeatedly receiving identical sensing queries, thereby saving a considerable amount of energy in the mobile devices. Before being reported to a corresponding crowdsensing server, $P_{in}^{(i)}$ is successively downsampled and compressed to reduce its size from γ_{in} to γ_{init}.

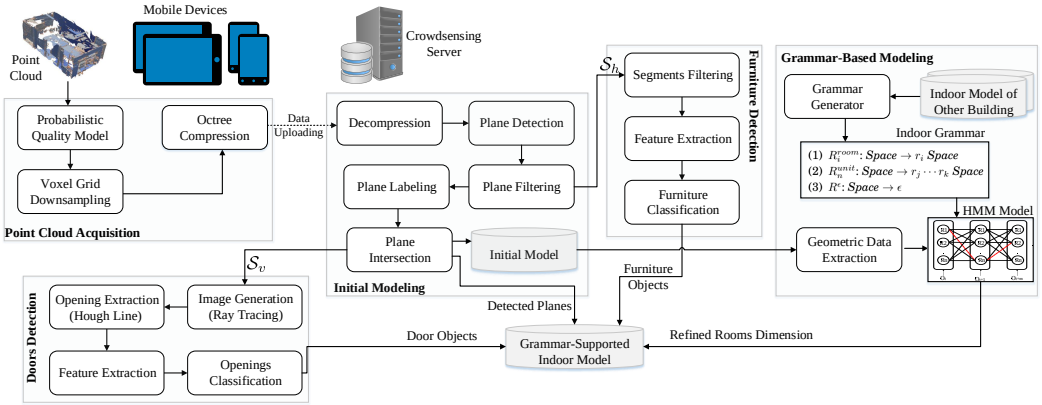


Fig. 2. MapSense processing pipeline at both the mobile devices and the crowdsensing server(s)

A crowdsensing back-end server is primarily responsible for sending sensing queries to the mobile devices as well as for processing the received point clouds $\mathcal{P} = P^{(i)}, P^{(i+1)}, \cdots, P^{(k)}$ to construct

detailed indoor models. To this end, the back-end server implements four main components, namely *room surface extraction*, *grammar-based modeling*, *doors detection*, and *furniture detection*. In the *room surface extraction* component, an initial indoor model is generated, before being refined using the structural knowledge encoded in an indoor grammar. As depicted in Figure 2, the point cloud $P_{init}^{(i)}$ is decomposed into several vertical and horizontal planar surfaces using the randomized Hough transform [34], before filtering out the noisy and redundant segments. Subsequently, the plane labeling step is employed to infer the type of obtained segments, i.e. wall, ceiling, or floor. To determine the relationship between these segments, the plane intersection step leverages the Cramer's rule [32] for estimating the intersection points between the walls, ceiling, and floor segments. These processing steps are performed on all the collected point clouds, thus generating an *initial* indoor model.

In fact, the initial models may suffer from inaccuracies due to relying on data collected by relatively low-quality sensors embedded in mobile devices. In general, point clouds captured by mobile devices are less accurate than those obtained by laser scanners [27]. Moreover, point clouds are mostly collected by non-experts. Accordingly, the various segments maybe tilted and empty spaces may exist in the point clouds. Therefore, the *grammar-based modeling* component is mainly devised to improve the modeling accuracy of the initial indoor model through incorporating design-time structural knowledge. A formal indoor grammar defines constraints, for instance, for the size of rooms, and is tailored to individual building types. Based on the information contained in the grammar, a hidden Markov model (HMM) is designed to predict the correct room type given observations extracted from the initial indoor model.

Aside from detecting the skeleton of the indoor model, it is also necessary to infer the doors and furniture objects. Both objects represent *context information* which is highly required in many indoor LBSs. For instance in exhibition centers, the indoor areas are highly dynamic thanks to the movable walls and furniture. In this case, modeling the available furniture is necessary to prevent the users from traversing a blocked hallway. Similarly for indoor virtual reality applications, it is crucial to identify all objects in the environment to achieve a proper integration between real and virtual worlds. In MapSense, we exploit the vertical and horizontal segments—detected in the previous component— to detect such context information. Specifically, the wall segments are forwarded to the *doors detection* component which converts each segment into a 2D image. Subsequently, the ray tracing algorithm [29] together with a random forest classifier [9] are utilized to cooperatively detect the door objects in the wall images. Similarly, the horizontal planes are utilized as an input to the *furniture detection* component. The main objective, in this step, is to correctly annotate the horizontal segments with semantic identifiers, e.g. chairs, tables, or noise objects.

In this article, we have some assumptions which are definitely valid in many public buildings, e.g. offices, hospitals, and campuses. First, we assume that the scanned areas have always *at least* four walls, a ceiling, and a floor. All surfaces are also assumed to be regular with flat shapes. In order to further reduce the computation time, the dimensions of high-level objects, e.g doors and furniture, are constrained by standard predefined values. In other words, several standard values for the dimension of each object are considered during the inference process. Accordingly, we can avoid redundant and unnecessary computations through reducing the search space. Finally, we assume that the indoor grammar is provided by a grammar generator. Such a grammar generator creates an indoor grammar by parsing a known indoor map of another floor of the same building or an indoor map of another building that has a similar architectural style. The detailed implementation of this grammar generator can be found in [6] and is beyond the scope of this work. Below, we introduce the concepts behind each component in the pipeline in more detail.

3 POINT CLOUD ACQUISITION

In this section, we introduce our approach for sampling point clouds from a set of depth sensors. Subsequently, we discuss collecting point clouds from the crowd in an energy-efficient manner. To this end, we introduce our probabilistic model for assessing the quality of the sampled point clouds before compressing and uploading them to the crowd-sensing servers

3.1 Point Clouds Generation

After receiving a sensing query from a crowdsensing server, the mobile owner reacts by scanning the requested area. The result of such a scan is a *registered* point cloud containing information about the structural objects, such as the walls, ceiling, floor, doors, and furniture. To generate registered point clouds, we harness the main features of Google Tango devices[1], namely *motion tracking*, *area learning*, and *depth perception* [26]. Motion tracking generally enables the mobile devices from tracking their own position and orientation in space, i.e. their pose, through tracking features in a set of image frames collected using a motion-tracking camera. Afterward, the position changes are integrated with the rotation and acceleration changes coming from inertial motion sensors. In fact, the motion traces, obtained by motion tracking, often suffer from accumulative drifts. To rectify these drifts, we integrate motion tracking and area learning features of the Tango technology. Specifically, area learning stores the key visual landmarks of a physical space in highly-compressed files, namely area description files (ADFs) for later identification of that space [20]. Such visual landmarks are exploited as *reference frames* to be matched with the frames collected while moving around the same space. In other words, when a Tango device learns a certain area, it estimates the instantaneous position of the device relative to a base frame, which is fetched from the ADF model.

The depth sensors generally generate several snapshots while scanning a certain space, e.g. room or room unit. Each snapshot comprises the 3D coordinates for all points in the scene. These coordinates use the coordinate frame of the depth-sensing camera as the origin. In MapSense, we perform two steps to turn the captured 3D points into a registered point cloud, including *point cloud registration* and *snapshots alignment*. In the former step, we integrate area learning and depth perception so that all sampled point clouds are referenced to a unique coordinate system. In a registered points cloud, each 3D point is identified with its location relative to a reference point in the building. This implies that the captured 3D points are not anymore estimated relative to the focal center of the depth camera, instead, they are computed relative to a stored ADF file. In the snapshots alignment step, we correct the orientation and position of the sampled snapshots. In general, the coordinate system of the range data from a tango device is relative to the depth sensor itself. Accordingly, if we capture consecutive snapshots while moving, the snapshots will not display correctly relative to each other. To properly align the snapshots, we leverage the motion traces—recorded while scanning the snapshots—to rotate and translate the snapshots into the same coordinate system.

3.2 Probabilistic Quality Model

After capturing a registered point cloud, MapSense adopts a quality checkpoint to prevent repeating identical sensing query due to reporting low-quality point clouds. In fact, point clouds, collected by non-experts using mobile devices, suffer from low quality, i.e. being incomplete, inaccurate, and/or oblique. The crowdsensing servers may repeat the sensing query once they receive extremely low-quality point clouds. In this manner, the energy overhead for scanning a certain area can be significantly increased. To avoid these situations, MapSense leverages a probabilistic quality

[1]Throughout this article, we focus on Tango devices while developing the proposed concepts. However, it is important to mention that these concepts are platform-agnostic, and can be easily applied to other platforms.

model to reject/accept the captured point clouds before being preprocessed and reported to the corresponding servers. In this regard, we define an *acceptance probability* P_{ac} as a conditional probability of the depth sensor to generate a high-quality point cloud given a set of quality metrics $\mathcal{X} = \{x_1, x_2, \cdots\}$. If the probability P_{ac} is found to be relatively low, then the crowdsensing App asks the user to rescan the same area.

3.2.1 Quality Factors. Before delving into the probabilistic quality model, we first analyze the factors which affect the quality of point clouds. First, the *acquisition system* and the measurement principle, i.e. time-of-flight, stereo matching, or structured light, strongly affect the accuracy of the depth data. In fact, point clouds generated from laser scanners are relatively more accurate than those acquired by mobile devices [27]. Despite being relatively inaccurate, MapSense relies on mobile devices to share the load of collecting point clouds among several users. Additionally, MapSense compensates for these inaccuracies through imposing the mobile device-generated point clouds to several filtering and refinement steps (cf. Section 4). Second, the *environmental conditions* such as variations in ambient light, pressure, temperature or humidity may have a non-negligible influence during the sampling process [31]. Since they rely on IR light, the mobile devices are not mostly capable of collecting depth information in areas illuminated by light sources high in IR such as sunlight or incandescent bulbs. Furthermore, lighting may negatively affect the localization accuracy. According to [5], the sampled rooms may shift from their actual locations whenever the lighting conditions are dissimilar to those conditions while recording the area learning model. To avoid such drifts, MapSense utilizes several ADF models recorded at different times of the day. While answering a sensing query, a participating mobile device selects an ADF model whose recording time is relatively close to the moment of scanning the queried area.

Third, the *characteristics* of the observed scene can strongly affect light reflection on the object surfaces. Examples of these characteristics are the object materials, surface reflectivity, and surface roughness. In fact, the mobile devices can hardly collect depth information from objects that do not reflect IR light, e.g. glass walls and windows [5]. Fourth, the *scanning geometry* broadly impacts the density and accuracy of the collected points. In this context, the scanning geometry is defined as the distance and orientation of scanned surfaces with respect to the involved depth sensor. Current mobile devices typically have an operating range between half a meter to four meters. Scanning an object at a distance outside this range leads either to collecting an insufficient number of points or to crashing the Tango core. Similarly, swift movements of the mobile devices can negatively contribute to the collected data quality.

3.2.2 Quality Metrics. Obviously, both of the characteristics of the observed scene and the scanning geometry can severely affect the quality of the captured point clouds. To measure their effect, MapSense considers two distinct quality metrics, namely *incompleteness* and *skewness*.

Incompleteness. As clarified in Section 3.1, each point cloud is composed of a set of snapshots which are to be aligned together. Through our experiments, we found that the number of points in each snapshot is highly sensitive to the scanning geometry and the type of the scanned surface. Specifically, the number of captured 3D points is inversely proportional to the distance between the mobile device and the targeted surface. Furthermore, the number of points per snapshot highly reduces when scanning a window, or a glass wall. To collect as many points in these scenarios, a user has to slowly scan these surfaces several times to cover as many details as possible. In MapSense, we define a points threshold α_p to quantify the incompleteness metric. The snapshots whose number of points n_s is smaller than the threshold α_p are annotated as *incomplete* snapshots S_{in}. Throughout our experiments, we found that setting α_p to five thousand 3D points enables proper annotation of snapshots. Accordingly, we define the probability of a point cloud to be

incomplete $P_{in} = \frac{|S_{in}|}{|S|}, \forall\, S_{in} : n_s \leq \alpha_p$ where $|S|$ is the total number of snapshots in the captured point cloud and $|S_{in}|$ is the number of incomplete snapshots.

Skewness. In the context of crowdsensing point clouds, skewness denotes the incorrect orientation of the scanned objects as a result of tilting the mobile devices while scanning an area. Indeed, it is hard to completely avoid skews emerged from improper scanning. Therefore, the probabilistic quality model is highly beneficial for rejecting severe cases in which the tilting of the scanned objects is broadly intolerable. To this end, the quality model leverages pose information, i.e. motion traces, to estimate the Euler's rotation angle, namely the roll angle ψ (cf. Equation 1). Specifically, the mobile devices provide the orientation data as a *quaternion* of the target frame with reference to the base frame [5]. These quaternions are typically defined in the form: $a + b \cdot \overrightarrow{i} + c \cdot \overrightarrow{j} + d \cdot \overrightarrow{k}$ where a, b, c, and d are real numbers while $\overrightarrow{i}$, $\overrightarrow{j}$, and $\overrightarrow{k}$ represent the standard orthogonal basis of the 3D space [8]. To quantify the skewness of captured point clouds, we map the roll angle ψ onto a random variable X_ψ which is normally distributed, i.e. $X_\psi \sim \mathcal{N}(\mu, \sigma^2)$. It is reasonable to set the mean μ to 90° which reflects the correct handling of the mobile devices while scanning point clouds. A variance $\sigma^2 = 30°$ has been found convenient for detecting intolerable skews. Accordingly, skewness probability P_{sk} can be estimated as expressed by Equation 2.

$$\psi = \text{atan2}(2(ab + cd), 1 - 2(b^2 + c^2)) \tag{1}$$

$$P_{sk} = 1 - P(X_\psi = \psi) \tag{2}$$

After defining the incompleteness probability P_{in} and skewness probability P_{sk} for the point cloud P_q, the acceptance probability P_{ac} is determined through fusing these probabilities. To this end, we employ the inclusion–exclusion principle. Specifically, we introduce a random variable $X_a \in \{0, 1\}$, which reflects the case that a captured point cloud P_q can be further preprocessed and uploaded ($X_a = 1$), or not ($X_a = 0$). Since our quality metrics are independent, we obtain

$$P_{sum} = P_{in}(P_q) + P_{sk}(P_q) - P_{in}(P_q) \times P_{sk}(P_q). \tag{3}$$

Accordingly, the acceptance probability P_{ac} is set to one, i.e. annotating P_q as a high-quality point cloud, if the quality requirement is achieved (cf. Equation 4). In this regard, we compare the probability P_{sum} to the quality requirement T_q encoded in the sensing query q.

$$P_{ac}(X_a = 1 \mid \mathcal{X}) = \begin{cases} 1 & \text{if } P_{sum} \leq T_q \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

3.3 Point Clouds Preprocessing

Once the captured point cloud P_q passes the quality checkpoint, it is subjected to downsampling and compression for the sake of reducing the communication energy overhead. At the outset, the point cloud P_q is downsampled for reducing the number of points representing the various structural objects. In general, a voxel grid is a set of tiny identical 3D boxes in space. The continuous geometric representation of a point cloud is mapped onto a set of voxels, i.e. $V = \{v_1, \cdots, v_i, \cdots, v_r\}$ where $r \ll m$, that best approximates the continuous representation. To downsample the point cloud, the points present in each 3D box v_i are approximated with their centroid $C := \frac{\sum_{i=1}^{\rho} p_i}{\rho}$ where ρ is the number of points in each voxel v_i. Accordingly, the resultant point cloud P_{vox} has a resolution of r, i.e. $P_{vox} \in \mathbb{R}^r$.

Finally, the point cloud P_{vox} is compressed to further reduce the communication energy overhead. To this end, we employ a lightweight algorithm, referred to as the Octree compression. Initially, an Octree is a tree-based data structure for processing 3D data where each internal node has exactly eight children. In Octree compression, the point cloud P_{vox} is spatially decomposed into an Octree

structure, before replacing the points present in each cell by the cell centers of the Octree's leaves. Subsequently, the resultant points are to be arithmetically encoded through considering only cells whose children are nonempty. The output of such a step is a lightweight version of the captured point cloud.

4 ROOM SURFACE EXTRACTION

The main goal of the *room surface extraction* component is to identify the various planar surfaces in a received point cloud $P_{init}^{(i)}$, before using them for inferring the structural objects, i.e. walls, ceiling, floor, furniture, and doors. To this end, we employ a two-phase process, including *initial model generation* and *grammar-based modeling*. In the first phase, we seek to infer the structural skeleton (i.e. walls, ceilings, and floors), before purifying the generated model from inaccuracies and incompleteness in the second phase. Below, we discuss both phases in more detail.

4.1 Initial Model Generation

As demonstrated in Figure 2, the initial model generation comprises four processing steps, including *plane detection*, *plane filtering*, *plane labeling*, and *plane intersection*. Figure 3 describes the various steps performed while generating initial indoor models. The plane detection step leverages the *randomized Hough transform* (RHT) algorithm to segment the point cloud $P_{init}^{(i)} \in \mathcal{P}$ into a set of planar surfaces $\mathcal{S}^{(i)} = s_1, s_2, \cdots, s_l$, where l is unknown a priori and its value may vary depending on the content of each point cloud (line 2). It is worth mentioning that the concepts discussed in this section are inspired by the extraction algorithm proposed in [3]. In general, the Hough transform is a computer vision technique which has traditionally been used for extracting parameterized objects, e.g. lines and circles, from images. Its core idea is to convert a set of points $p_1, p_2, \cdots, p_r$ in the Cartesian space into a set of lines in the Hough space.

Due to its intense computations, the standard Hough transform typically requires high computation time. To overcome this problem, several variants have been proposed. The RHT algorithm [34] is one of these variants which reduces the computation time through randomly selecting points from the point cloud. Hence, the RHT algorithm avoids mapping every point $p_i \in P_{init}^{(i)}$ onto a 3D plane in the Hough space. In fact, the intersection among solely three planes in the Hough space is sufficient for checking whether their corresponding points belong to the same plane in the Cartesian space. When several points vote for the same plane, i.e. the same cell, and the cell value exceeds an accumulation threshold γ_{cell}, those points are considered to be lying in the same plane. Subsequently, these points are removed from the point cloud to avoid reprocessing them. Since RHT may randomly select extremely far points, we deliberately restrict the random selection of points imposed by the RHT algorithm. To this end, the distance between the selected points has to lie in the range $[\beta_{min}, \beta_{max}]$. Accordingly, the selected points will mostly belong to the same plane while avoiding the generation of extremely small planes.

After identifying the various surface planes $s_1, s_2, \cdots, s_l$ in the point cloud $P_{init}^{(i)}$, the plane filtering step removes noisy and redundant planes, referred to as the clutter. To this end, we divide the obtained planes into two groups, namely vertical $\mathcal{S}_v$ and horizontal $\mathcal{S}_h$ planes (cf. lines 6 and 7 in Figure 3). For each regular room, we expect to optimally obtain four vertical planes, i.e. four walls, and two horizontal planes, i.e. ceiling and floor. In this step, we employ the *principal component analysis* (PCA) method [14] to determine the normal $n = (\vec{v}, \eta)$ of each surface plane. The eigenvector $\vec{v}$ serves as a direction, while the corresponding eigenvalue η denotes the variance in that direction. The vertical planes typically have circa zero eigenvalue in the y direction. Accordingly, we set a surface threshold ε_{sur} of the eigenvalues below which the plane is considered to be vertical.

Require: point clouds $\mathcal{P} = P_{init}^{(1)}, \cdots, P_{init}^{(k)}, \varepsilon_{sur}, \varepsilon_{wall}$

1: **for all** point clouds $P_{init}^{(i)} \in \mathcal{P}$ **do**
2: $\mathcal{S}^i = \boldsymbol{RHT}(P_{init}^{(i)}, \beta_{min}, \beta_{max})$ ▷ Plane detection
3: **for all** planes $s_j \in \mathcal{S}^i$ **do**
4: $\vec{v}_j, \eta_j \leftarrow \boldsymbol{normal}(s_j)$
5: **end for**
6: $\mathcal{S}_v \leftarrow \{s_j \in \mathcal{S}^i \mid \eta_{j,y} < \varepsilon_{sur}\}$
7: $\mathcal{S}_h \leftarrow \{s_j \in \mathcal{S}^i \mid 1 - \eta_{j,y}/\sqrt{\eta_{j,x}^2 + \eta_{j,y}^2 + \eta_{j,z}^2} < \varepsilon_{sur}\}$
8: $\mathcal{S}_{clutter} \leftarrow \{s_j \in \mathcal{S}^i \mid s_j \notin \mathcal{S}_v, \mathcal{S}_h\}$ ▷ Plane filtering
9: **for all** segments $s_j, s_{j+1} \in \mathcal{S}_v$ **do**
10: $\omega_{j,j+1} \leftarrow \boldsymbol{distance}(s_j, s_{j+1})$ ▷ Wall labeling
11: **end for**
12: $S_{wall} \leftarrow \{s_j \cup s_{j+1} \mid \omega_{j,j+1} \leq \varepsilon_{wall} \ \ \& \ \ \eta_j \times \eta_{j+1} \leq \varepsilon_{norm}\}$
13: **for all** segments $s_i, s_{i+1} \in \mathcal{S}_v$ **do**
14: $p_0 \leftarrow \boldsymbol{intersect}(s_j, s_{j+1}, s_{ceil})$ ▷ Plane intersection
15: **if** $p_0 == 0$ **then**
16: **repeat**
17: $\boldsymbol{extend}(s_j, s_{j+1}, s_{celing})$
18: **until** $p_0 > 0$
19: **end if**
20: **end for**
return the structural parameters $S_{wall}, s_{ceil}, s_{floor}, p_0$
21: **end for**

Fig. 3. Structural data extraction from the crowdsensed point clouds

Similarly, a plane is recognized as a horizontal one if it satisfies the following inequality [23]:

$$1 - \frac{\eta_y}{\sqrt{\eta_x^2 + \eta_y^2 + \eta_z^2}} < \varepsilon_{sur} \tag{5}$$

where η_x, η_y, η_z represents the eigenvalues in the x, y and z directions. In the next step, the detected horizontal and vertical planes are labeled as either wall, floor, or ceiling (cf. line 12 in Figure 3). To recognize the ceiling s_{ceil} and floor s_{floor} segments, we simply search for the horizontal segments with the highest and lowest y values, respectively. Equations 6 and 7 formalize these conditions where k_j is a predicate whose value is set to one only when the examined segment is horizontal.

$$s_{ceil} = s_i \mid y_i = \max_{j \in [1,l]} (y_j \times k_j) \tag{6}$$

$$s_{floor} = s_i \mid y_i = \min_{j \in [1,l]} (y_j \times k_j) \tag{7}$$

Subsequently, the vertical planes are to be examined to correctly detect the wall segments. In fact, wall detection is not a straightforward task owing to the noisy nature of the collected point clouds and the existence of furniture objects close to the walls. Hence, there may exist several vertical segments representing the same wall. In this case, we identify the wall segments through projecting all vertical planes onto the x-z plane. Afterward, we adopt the 2D α-shapes method [4] to find the concave hull of each projected segment. The concave hull represents an envelope surrounding the outer points in each segment. Figure 4 depicts the top view of two vertical segments s_i and

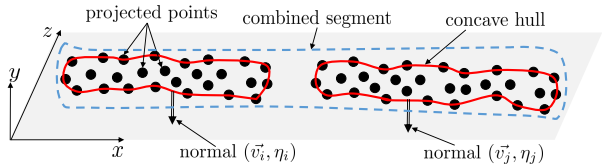


Fig. 4. Top view of two projected planes and their combined segment

s_j, where both segments have been projected on the x-z plane. After detecting the concave hull (i.e. red line in Figure 4) of each segment, we check for the relationship between the neighboring hulls. Specifically, any two segments s_i, s_j are considered belonging to a single wall, if (1) the distance between them is below a wall threshold ε_{wall} and (2) the cross product of their normals corresponds up to a certain value ε_{norm}. Once these conditions are satisfied, we can readily combine both segments into a single wall segment (cf. blue dotted line in Figure 4). Finally, the points of the combined hulls are averaged to inscribe a single line representing the wall surface. This process is repeated until all projected segments are examined.

So far, the detected objects, i.e. walls, ceiling, and floor, are not related to each other. The final step in the initial model generation phase is to find the intersection points between these objects (cf. line 14 in Figure 3). To this end, we solve a set of linear formulas—representing the intersected planes—using the Cramer's rule [13]. Generally, the Cramer's rule uses determinants to solve a system of linear equations with as many equations as unknowns. The main beauty behind this method is that it can selectively solve for solely the required parameters, i.e. not all unknown parameters, thus avoiding unnecessary computations. As illustrated in Figure 5, the intersection between two planes $P_1 : a_1x + b_1y + c_1z + d_1 = 0$ and $P_2 : a_2x + b_2y + c_2z + d_2 = 0$ can be defined as a straight line L. The direction of intersection is along the vector $\vec{u}$ that is the cross product of a vector normal n_1 to plane P_1 and a vector normal n_2 to plane P_2. If the two planes are parallel, their normal vectors will be also parallel which means that $\vec{u} = n_1 \times n_2 = 0$. Alternatively, if the two planes are not parallel, $\vec{u}$ is a direction vector for the intersection line L. In this case, the vector $\vec{u}$ is perpendicular to both n_1 and n_2 and thus is parallel to both planes, as depicted in Figure 5.

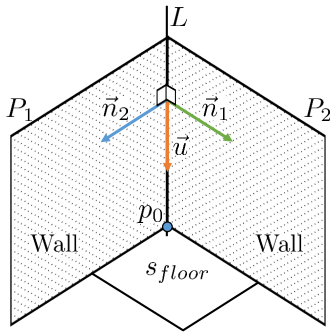


Fig. 5. Intersection of wall, floor, and ceiling segments

After detecting the intersection line L, it is necessary to determine the corner points which lie on the line L. To this end, we utilize a third plane $P_3 : a_3x + b_3y + c_3z + d_3 = 0$ perpendicular to the line L and have a normal vector $n_3 = \vec{u}$. Cramer's rule expresses the corner point p_0 in terms of the normal vectors of the three intersected planes P_1, P_2 and P_3 (cf. Equation 8). Specifically, we exploit the ceiling horizontal plane s_{ceil} to obtain the upper corner and the floor plan s_{floor} to get

the lower corner of the room. It is worth mentioning that the detected planes, i.e. walls, ceiling, or floor may be incomplete, hence they do not intersect with each other. In this case, we deliberately extend the planes in the four directions till an intersection line L is found and a nonzero value of p_0—in case of corner detection—is obtained (cf. line 18 in Figure 3).

$$p_0 = \frac{-d_1(n_2 \times n_3) - d_2(n_3 \times n_1) - d_3(n_1 \times n_2)}{n_1 \cdot (n_2 \times n_3)} \tag{8}$$

The output of the initial model generation phase is a skeleton of the floor plan, i.e. a set of intersected segments representing the walls, ceilings, and floors. As it will be clarified in Section 7, the initial model suffers from inaccuracies due to collecting data using *commercial* mobile devices carried by non-experts. In the next section, we elaborate on the grammar-based refinement approach to generate highly-accurate indoor models.

4.2 Grammar-Based Modeling

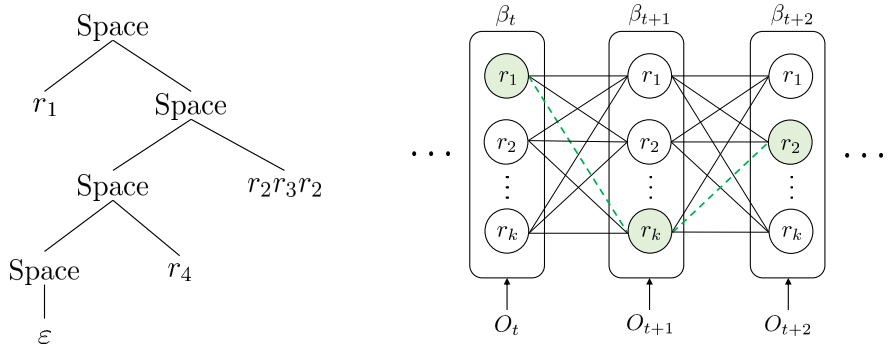
In this section, we introduce our formal grammar exploited for enhancing the modeling accuracy. We start with defining formal indoor grammars which encode structural information. Subsequently, we present a hidden Markov model through which a precise indoor model can be generated using the indoor grammar.

4.2.1 Indoor Grammars. Generally, public buildings are composed of hallway spaces traversed by individuals to reach various rooms. The size and relative ordering of such rooms typically follow architectural principles and semantic relationships. In this context, formal grammars represent an effective tool for embedding such a priori structural knowledge during the design time. Formally, an indoor grammar is defined as a quadruple $G = (\upsilon_n, \upsilon_t, s, r_i)$, where $\upsilon_n = Space$ is the set of non-terminal symbols, $\upsilon_t = \{\epsilon, r_a, r_b, r_c, \cdots\}$ is the set of terminal symbols, r_i is the set of production rules and $s = Space$ is the axiom. The axiom represents an empty non-hallway space which is to be further divided into rooms. The terminals υ_t describe solids that are not divisible any further.

Each terminal symbol $r_i \in \upsilon_t$ constitutes a class i of rooms. Both non-terminals and terminals have attributes such as the space's geometric extent and probability. The production rules r_i are defined as replacement rules that perform a split to divide a *Space* into two *Space* elements along a partitioned plane. For instance, Figure 6a demonstrates the rule derivation process where a *Space* element is to be iteratively split into a set of rooms, e.g. r_i, or room units, e.g. $r_2 r_3 r_2$. Obviously, the iterations stop when there exist no further empty spaces. Such a case is expressed by the symbol ε, as depicted in Figure 6a. To define an indoor grammar, we developed in [6] a tool—based on iterative search—to instantiate grammar rules from a small set of motion traces.

Although the indoor grammar encodes knowledge about the geometrical semantics, it lacks knowledge of the neighborhood of these semantics. Hence, we define two probability models for the grammar: (1) the *a priori* probability $P_a(r_i)$ principally stands for the relative frequency of occurrence of a room or room unit and (2) the *relationship* probability $P_r(r_j \mid r_i)$ is a conditional probability which models the relationship between rooms or room units. The room units are defined in a similar manner as a sequence of rooms. Below, we explain our proposed method for enhancing the initial indoor model using the described indoor grammar.

4.2.2 Indoor Model Generation. In this article, we employ the *split grammar* [33] to generate an accurate representation of the room layout. In particular, the split grammar is used as a model fitting tool. Specifically, the production rules have to be applied along a given line, referred to as the reference line segment (RLS). These RLS lines $\mathcal{L} = L_1, \cdots, L_u$ are used as a reference while cascading the successive rooms $r_1, r_2, \cdots$ (cf. Figure 7). To find the sequence of rooms for each RLS line, we first extract the dimensions and positions of the structural objects from the initial model.



(a) Grammar rules derivation (b) HMM trellis for room layout generation

Fig. 6. Grammar-based indoor modeling

Table 1. Two grammar rules representing a room and a room unit

	R_1^{room}	R_5^{unit}
Rule	$Space \rightarrow r_1 Space$	$Space \rightarrow r_3 r_2 r_3 Space$
Width	2.4 m	19.2 m
A-priori	0.06	0.04
Type	small office	two executives with assistant's office

Subsequently, we utilize this structural data as an input to an *Hidden Markov Model* (HMM) to properly estimate the room layout. In general, deriving indoor models using formal grammars can be formulated as Markov chain Monte Carlo (MCMC) problem (cf. [30]). In particular, the MCMC problem employs a set of probability distributions to generate several floor plans with random room sequences. In this case, no real observations, e.g. motion traces or point clouds, are needed. To reduce the modeling error, we proposed in [24] an indoor modeling approach which sequentially generates several hypotheses of the floor plan using a constrained random walk on an MCMC model. Afterward, a hypothesis—which minimizes the modeling error—is selected as the indoor map where the error is estimated using a set of motion traces.

Alternatively, MapSense formulates the problem of finding the probable room sequence for each RLS line as an HMM problem. The intuition behind using HMM models is enabling us from directly deriving indoor maps rather than selecting one from a set of hypotheses. To this end, an HMM model exploits the extracted geometrical information while deriving the indoor map to adjust the probability distributions. As it can be seen in Figure 6b, each space β_i represents an HMM state which has several rule candidates $r_i, \cdots, r_k$. The Viterbi algorithm determines the most probable path between the various states which is corresponding to the rooms layout. As depicted in Figure 6b, the green nodes denote the inferred sequence of grammar rules, i.e. the room layout comprises room r_i, room r_k, and then room r_2.

Knowledge Extraction. To extract the structural knowledge from the initial model, we harness a set of RLS lines, provided by the indoor grammar (cf. [6]). For each RLS line $L_k \in \mathcal{L}$, we iterate over the set of line segments — existing in the initial indoor model — and then search for branches. A branch represents a wall segment where the distance between two consecutive branches is used as the room's size. In order for a line segment to be recognized as a branch of the current RLS line,

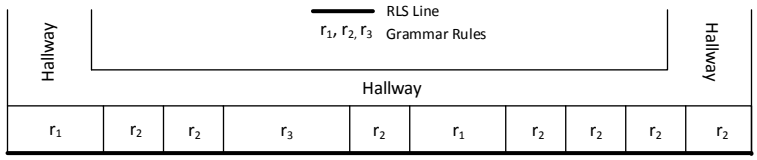


Fig. 7. Example of assigning a room sequence along a RLS line

it has to satisfy two conditions: (1) being orthogonally oriented to the RLS line, and (2) intersecting with the RLS line or having an endpoint located in the vicinity of the RLS line. The branches which overshoot or intersect with an RLS line are recognized without problems. However, some potential branches may not be connected to an RLS line due to incomplete sampling data. Hence, we employ a distance threshold as a safety margin to avoid overlooking possible branches.

HMM-Based Model Derivation. MapSense formulates the problem of finding the probable room sequence for each RLS line as a *hidden Markov model* (HMM) [25] where the best path between the various states determines the room sequence. To this end, the HMM model uses as an input the extracted geometrical information while deriving the indoor map to adjust the probability distributions. An HMM model is generally a statistical model in which the system being modeled is assumed to be a Markov process with unknown states, and the challenge is to determine the hidden states $\mathcal{B} = \beta_i \mid i = 1, 2 \cdots, n$ from a set of observables $O = o_j \mid j = 1, 2 \cdots, n$. Each *hidden state* β_i emits an observable, whose likelihood is given by a conditional emission probability $P_e(O_t \mid q_t = \beta_i)$ where q_t denotes the current hidden state. An HMM model $\Gamma := \langle \mathcal{B}, O, \pi, P_t, P_e \rangle$ also permits transitions among its hidden states, where π is the vector of initial state probabilities. These transitions are governed by a different set of likelihoods called transition probabilities $P_t(q_{t+1} = \beta_j \mid q_t = \beta_i)$. In our scenario, the rooms $r_1, r_2, r_3, \cdots, r_k$ represent the hidden states, while the extracted geometrical information from the initial model denotes the observables (cf. Figure 6b). The number of hidden states n in each RLS line L_k depends on the number of extracted observations. The initial probability vector π at $t = 0$ is set to the a priori probabilities, $\pi(r_i) = P_a(r_i)$. Whereas, the transition probability from rule r_i to rule r_j is given by

$$P_t(r_j \mid r_i) = \frac{P_a(r_j)}{P_a(r_i)} \cdot \frac{P_r(r_i \mid r_j)}{P_r(r_j \mid r_i)}. \tag{9}$$

The emission probability P_e captures the likelihood of observing a certain room size given the hidden states Ψ. To estimate the emission probabilities, we map each rule r_i onto a random variable X_{r_i} which is normally distributed, i.e. $X_{r_i} \sim \mathcal{N}(\mu_i, \sigma^2)$. The mean μ_i is set to the width of the grammar rule, i.e. $\mu_i = W_i$, where a variance σ of one meter is found to be sufficient for properly inferring the rules. Consequently, the emission function $P_e(O_t \mid q_t = r_i)$ can be simply estimated from the probability density function of the rule random variable X_{r_i} as follows

$$P_e(O_t \mid q_t = \psi_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp[\frac{-1}{2\sigma^2} \cdot (O_t - W_i)^2]. \tag{10}$$

Figure 8 depicts the HMM model-based rule sequence inference algorithm. The sequence of observations is defined as the extracted room size from the initial indoor model. These semantics have to be filtered out before being applied to the HMM model (line 2). In particular, there may exist two neighboring line segments s_i, s_{i+1} representing the same wall, but they have been scanned from different sides. Accordingly, the Euclidean distance $\delta(s_i, s_{i+1})$ between these segments is deliberately discarded given that the distance δ is less than a threshold τ (lines 3-9). After defining

Require: grammar rules $r_i \in T$, line segments from the initial model $\mathcal{S}$, RLS lines $\mathcal{L}$, threshold τ

1: **for all** RLS line $L_k \in \mathcal{L}$ **do**
2: $\mathbb{B} \leftarrow$ orthogonal lines $s_i \in \mathcal{S}$ with minimum distance to the RLS line L_k
3: **for all** line $b_i \in \mathbb{B}$ **do**
4: **if** $\delta(b_i, b_{i+1}) > \tau$ **then**
5: $O_t \leftarrow \delta(b_i, b_{i+1})$, where $O_t \in O$
6: **else**
7: **skip** the line b_i ▷ lines filtering
8: **end if**
9: **end for**
10: **compute** the probabilities $P_e(O_t \mid r_i), \quad r_i \in T$
11: $\Gamma \leftarrow \Psi, O, \pi, P_t, P_e$ ▷ HMM modeling
12: **for all** $\psi_t \in \Psi$ **do** ▷ modified Viterbi
13: **for all** $r_i \in \psi_t$ **do**
14: **for all** $r_j \in \psi_{t-1}$ **do**
15: **if** $W_i^t + W_j^{t-1} \leq \overline{L}_k$ **then**
16: $\alpha_i^j = \eta_{t-1}(r_j) \cdot P_t(r_i \mid r_j) \cdot P_e(O_t \mid r_i)$
17: **else**
18: **skip** the path $r_j \rightarrow r_i$
19: **end if**
20: **end for**
21: $\eta_t(r_i) \leftarrow \max_j \left(\alpha_i^j\right)$
22: $W_i^t \leftarrow W_i^t + W_j^{t-1}$
23: **end for**
24: **end for**
25: **estimate** ϕ to get the optimal path ▷ backtracking
26: **update** the probabilities π and P_t
27: **end for**

Fig. 8. HMM-based rule sequence inference algorithm

the HMM model, we can now find the best path (i.e. a sequence of rules for each RLS line) which has the maximum joint probability (plotted as a red dashed line in Figure 6b). To this end, we employ the Viterbi algorithm [10] to compute the best path through the HMM trellis. The Viterbi algorithm utilizes dynamic programming to quickly find the path through the trellis that maximizes the product of the emission probabilities and transition probabilities. For each intermediate and terminating state $r_i \in \psi_t$, there exists a most probable path to that state, referred to as the *partial* best path. Accordingly, the probability of the partial best path to a state r_i is recursively obtained by

$$\eta_t(r_i) = \max_j \left(\eta_{t-1}(r_j) \cdot P_t(r_i \mid r_j) \cdot P_e(O_t \mid r_i)\right). \tag{11}$$

To find the overall best path, the Viterbi algorithm utilizes a back pointer ϕ to the predecessor that optimally provokes the current state, where $\phi_t(i) = \arg\max_j \left(\eta_{t-1}(j) \cdot P_t(r_i \mid r_j)\right)$. To simplify the process of searching the best sequence, we filter out paths whose accumulative length is greater than the length of the corresponding RLS line (lines 15-19). The output of the room surface extraction component is an accurate representation of the floor plan. In the next section, we use the wall segments to infer possible door objects.

5 DOORS DETECTION

The doors detection component comprises several processing steps, including *image generation*, *possible opening extraction*, and *classification*. At the outset, the Euclidean distance between each point in the wall segment $s_{wall} \in \mathcal{S}$ and the vertical x-y plane is individually estimated and then stored in a depth map. In addition to these depth maps, each wall segment s_{wall} is harnessed to generate a 2D *label* image whose pixels represent the *occupied*, *empty*, or *occluded* spaces in s_{wall} (cf. Figure 9). In this context, occlusion refers to the effect of one object, e.g. furniture, in a 3D space blocking another object, e.g. wall, from view. The depth images are used as an input to a hybrid edge detection mechanism, a combination of the Canny detector and the Sobel filter [21], to determine the bounding box of the empty areas, i.e. door candidates. Once the door candidates are defined, the random forest algorithm together with K-Means clustering are exploited to: (1) infer whether these door candidates represent actual doors and (2) overcome the redundancy problem in which several candidates exist for a single door object. To this end, several features, e.g. percentage of occluded and empty areas, are to be extracted from the depth and label images.

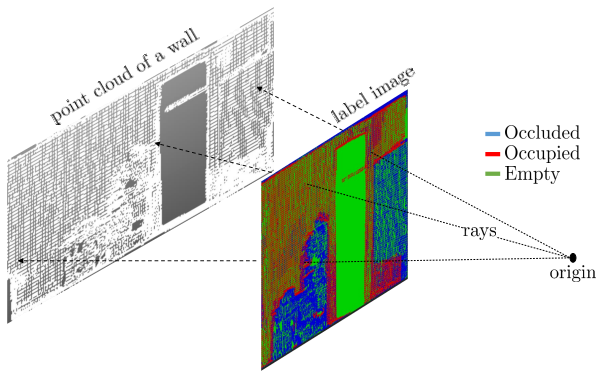


Fig. 9. Ray tracing for a wall surface

In the image generation step, we employ the well-known *ray tracing* algorithm [7] to render the label and depth maps of the wall segments (cf. Figure 9). The core idea behind ray tracing is to search for intersections between a set of rays—fired from an origin point—and the points inside the segment s_{wall}. When there is a hit with a distance threshold d, the discrete point p_i is labeled as "occupied". If the hit occurs with a point that has a distance less than d from the origin, the point is labeled as "occluded" and a corresponding pixel is generated in the depth image. The "empty" label is assigned to a point p_i when the ray is not interrupted throughout the trace, i.e. there is no obstacle between the origin and destination points. For the empty label, the distance threshold is modified to $d + d_0$ where d_0 is a safety margin used to make sure that the discrete point is definitely not occupied or occluded. After scanning all the points belonging to the wall surface, we obtain label and depth images as depicted in Figure 9. The ray tracing is performed individually for each wall surface, hence multiple label and depth images will be generated.

The second processing step of the doors detection algorithm is to detect the bounding box of every empty area, i.e. door candidates. To this end, we first convert the depth images into edge images, before detecting such edges using the probabilistic Hough line algorithm. For each depth image, the *Canny edge detection* algorithm is employed to generate an edge image I_{Canny} which includes all horizontal and vertical lines. In fact, the canny detection algorithm provides a poor estimation of the various lines. To enhance the detection accuracy, we additionally use a second edge detection algorithm, referred to as the *Sobel* filter. The Sobel filter is an approximated method

for determining the derivative of an image in a certain direction. Accordingly, it generates two edge images: I_h in the horizontal direction and I_v in the vertical direction. The fusion of I_{Canny}, I_h, and I_v results in a more accurate edge image I_{com}. Finally, the image I_{com} is used as an input to the Hough line algorithm to estimate the vertical and horizontal lines. For every two vertical and two horizontal lines, all possible rectangular boxes will be constructed which are considered as door candidates.

So far, we obtained a set of rectangular boxes which may represent either actual door objects or improperly scanned areas. The next step of the doors detection component involves inferring the type of such boxes. To this end, we examined two machine learning classifiers, namely support vector machine (SVM) and random forest [9]. Due to performing poorly compared to the random forest method, the SVM classifier is skipped from this discussion. It is worth mentioning that relatively small boxes are excluded from the classification step to further improve the detection accuracy. There exist two stages in the random forest algorithm, namely forest creation and prediction. To create the random forest model, it is necessary to extract a set of features from the obtained rectangular boxes and the label images. Figure 10 demonstrates a wall surface containing a door candidate (shown in green). As it can be seen in the figure, eight features related to the dimensions and position of the door candidate and the wall surface can be extracted. Furthermore, we can extract the percentage of the occupied, occluded and empty areas from the label images. These features are used by all the decision trees, however, each tree randomly receives different training data set.

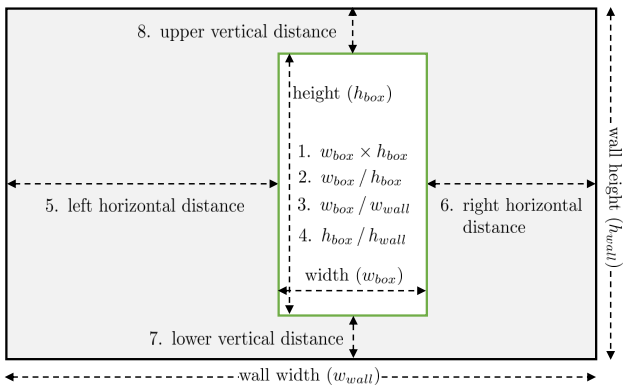


Fig. 10. Doors classification features

In the prediction phase, the classifier is given the features of a set of door candidates as an input. Each decision tree in the forest generates an outcome, before calculating the votes for each predicted outcome. The highly-voted predicted outcome is considered as the final prediction. After classification, there may exist several redundant potential doors, i.e. different classified doors may represent the same door object (cf. evaluations in Section 7). To resolve this situation, we employ the *K-Means clustering* method to divide the redundant door objects into k clusters where each door object is denoted by its absolute area and its center. Specifically, the clusters are to defined according to the empty, occupied, and occluded areas. For instance, the objects which have nearly similar empty space are grouped together. In each cluster, we average the area values of the door objects in that cluster to define a merged door object. Finally, we select the cluster which has the largest empty area and smallest occupied and occluded area as the most accurate door object.

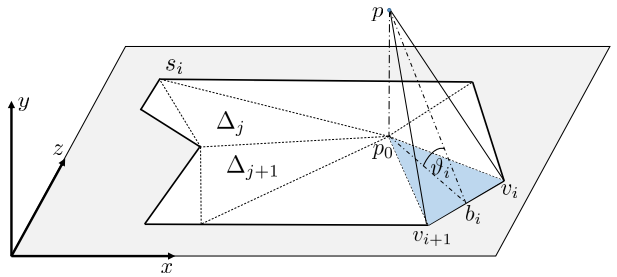


Fig. 11. Area of a potential furniture object

6 FURNITURE DETECTION

In the furniture detection component, the horizontal segments $\mathcal{S}_t \subset \mathcal{S}_h, y(s_i) \notin \{y_{min}, y_{max}\}$ are taken as an input to a random forest classifier to infer a set of furniture objects, e.g. tables and chairs. Before delving into the classification step, the horizontal segments are to be filtered to remove the surfaces: (1) whose area is equal or larger than the ceiling and floor surfaces and (2) whose distance to the floor plane is below a furniture threshold γ_{fur}. As an input to the random forest classifier, we extract five features from the remaining horizontal segments, i.e. potential furniture objects $\mathcal{S}_{fur} \subset \mathcal{S}_t$. The extracted features comprise the absolute area A_{s_i}, the distance D_{s_i} between a potential furniture object s_i and the floor segment s_{floor}, and the width w_{s_i} and height h_{s_i} of the object s_i. In fact, the horizontal segments—represented by the concave hulls—are typically irregular. In this case, the absolute area A_{s_i} of a potential furniture object s_i can be determined through decomposing it into a set of triangles $\Delta_1, \cdots, \Delta_n, ,$ as depicted in Figure 11.

Consider a triangle Δ_i (blue in Figure 11) whose vertices are v_i, v_{i+1}, and p_0. The area A_{Δ_i} of such a triangle is given by $\frac{1}{2}|v_i v_{i+1}|\,|p_0 b_i| = \frac{1}{2}|v_i v_{i+1}|\,|pb_i| \cos\vartheta_i$, where p is an arbitrary point in the space. Accordingly, the total area A_{s_i} is directly computed by accumulating the areas of all triangles, i.e. $\sum_{i=1}^{n} A_{\Delta_i}$. Similarly, the distance between a potential furniture object s_i and the floor plane s_{floor} can be estimated as the distance between two parallel planes. Equation 12 expresses the distance D_{s_i} between the object s_i and the floor plane using the Hesse Normal form, where a, b, c and d are the furniture object's coefficients while x_1, y_1, z_1 represents a 3D point on the floor plane s_{floor}.

$$D_{s_i} = \frac{|ax_1 + by_1 + cz_1 - d|}{\sqrt{a^2 + b^2 + c^2}} \tag{12}$$

Finally, the width w_{s_i} and height h_{s_i} of a potential furniture object s_i are estimated through turning the irregular plane s_i into a regular one. To this end, we determine the extreme points, i.e. $x_i, y_i, z_i \forall i \in \{\min, \max\}$, before inscribing a bounding box that encloses these points. After defining the features, the random forest classifier is trained using an Online dataset of the dimensions of various tables and chairs. The output of this component is a set of planes representing either tables or chairs. After successfully inferring all structural objects, we combine them together to construct a detailed and accurate 3D indoor model.

7 PERFORMANCE EVALUATION

In this section, we evaluate the effectiveness of our proposed pipeline in efficiently generating precise indoor models. We first describe the setup of our evaluations. Afterward, we discuss the

results of each processing step, before discussing the QoS metrics in terms of modeling accuracy, execution time, and energy efficiency of the mobile devices.

7.1 Experimental Setup

To test our implementation, we collected more than 3 GBytes of data, i.e. 105 point clouds, using Tango mobile devices which are equipped with a 120° front-facing camera, a 4 MP RGB-IR rear-facing camera, and a 170° motion tracking camera [5]. We started with designing an Android App for collecting, preprocessing, and reporting the point clouds to a back-end server. We performed our experiments in two different buildings, including the second floor of the IPVS[2] institute and the fourth floor of the ifp[3] institute, University of Stuttgart. Since the point clouds are to be collected by individuals, we have to examine the performance of MapSense using data collected by different volunteers. Therefore, each use case, i.e. the IPVS floor and the ifp floor, has been scanned by a different group of volunteers. Each volunteer was asked to scan a number of rooms several times while moving freely between the hallways and the different rooms. Nevertheless, each volunteer had to follow two main constraints: (1) slowly scanning the rooms and (2) keeping a distance of 0.5m between the mobile device and the surroundings. If these conditions are violated, the Tango core, which provides core functionality for Tango applications, swiftly crashes.

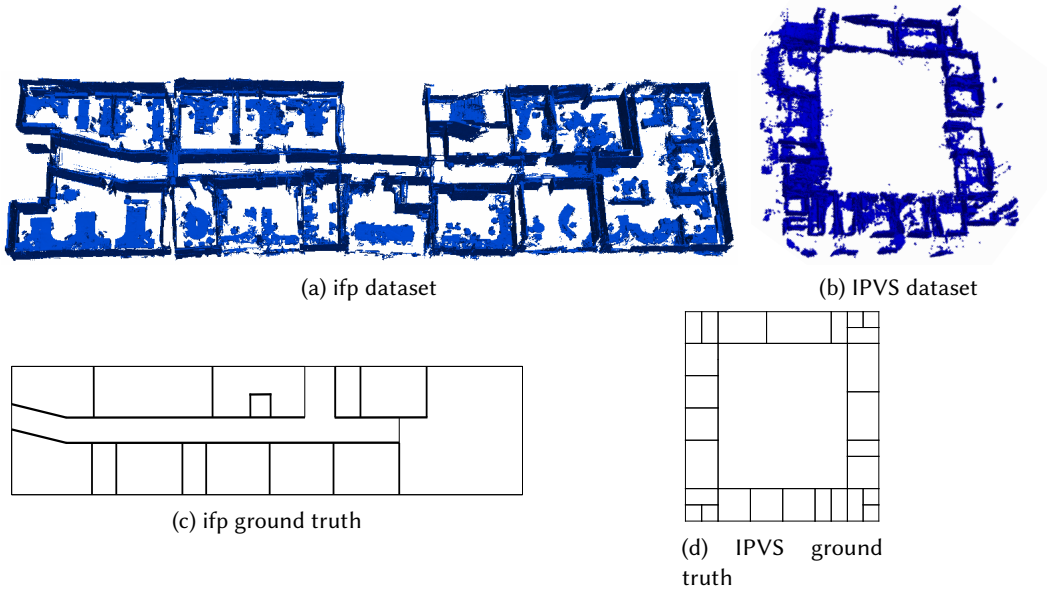
Figures 12a and 12b depict the combined point cloud representing the various rooms in the ifp and the IPVS datasets (the 2D ground truth representation of both floors are given in Figures 12c and 12d), noting that the ceiling layer has been removed from the ifp dataset to show the content of each room. To process the point clouds, we exploited already-existent functions from the point cloud library (PCL) and the 3DTK library. PCL is a standalone open-source project of C++ functions for processing point clouds [28]. We integrated some of these functions in our implementation, such as the voxel grid and Octree compression. Similarly, we incorporated RHT functions in our code from the 3DTK library. To implement the random forest classifier and ray tracing, we utilized some functions from the OpenCV library. Figure 13 summarizes the parameters and their values which have been used throughout the evaluations.

7.2 Model Generation

In this section, we examine the modeling accuracy of the initial and the grammar-supported indoor models developed using our proposed pipeline. We start with showing the results of each processing step, before comparing the accuracy of the initial and the grammar-based indoor models. Figure 14a visualizes a raw point cloud of a certain room before being processed by our processing pipeline. Figure 14b demonstrates the various planes detected in such a point cloud using the RHT algorithm. In this figure, a unique color is given to each plane to clearly show the detected planes. It is important to notice that the missing points, e.g. parts of the ceiling in Figure 14a, have been compensated through continuously extending the detected planes till they are intersected (cf. Section 4.1). Figure 14c depicts the same point cloud after filtering out the clutter and the noisy segments. The figure shows only the segments which have been classified as either horizontal or vertical planes. During the plane extraction step, we adapted the values of the RHT distance range where for small rooms—whose width is below 10m—the range has been set to [0.2m, 0.5m] and [0.5m, 5m] for larger rooms.

Figures 15a and 15c depict the initial models of the ifp floor and the IPVS floor generated from combining the walls detected in each room. In fact, the obtained model is relatively irregular and the rooms width is mostly inaccurate. Hence, we harness such an initial model to derive

[2]IPVS stands for institute for parallel and distributed systems
[3]ifp stands for institute for photogrammetry



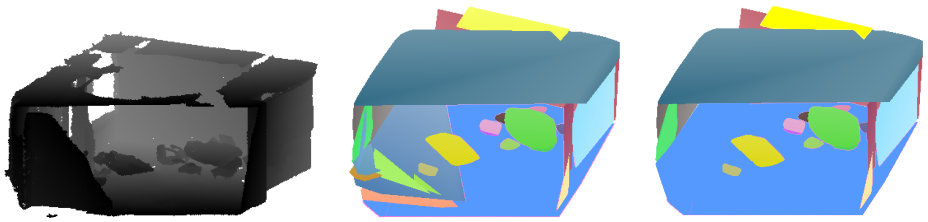
(a) ifp dataset (b) IPVS dataset

(c) ifp ground truth (d) IPVS ground truth

Fig. 12. Combined point clouds used in the evaluations

System Parameter	Value	System Parameter	Value
accumulation threshold γ_{cell}	20	RHT range $[\beta_{min}, \beta_{max}]$	[20 cm, 5 m]
surface threshold ε_{sur}	0.1 m	wall threshold ε_{wall}	0.2 m
furniture threshold γ_{fur}	0.7 m	normals threshold ε_{norm}	2 exp −1

Fig. 13. parameters used in the evaluations

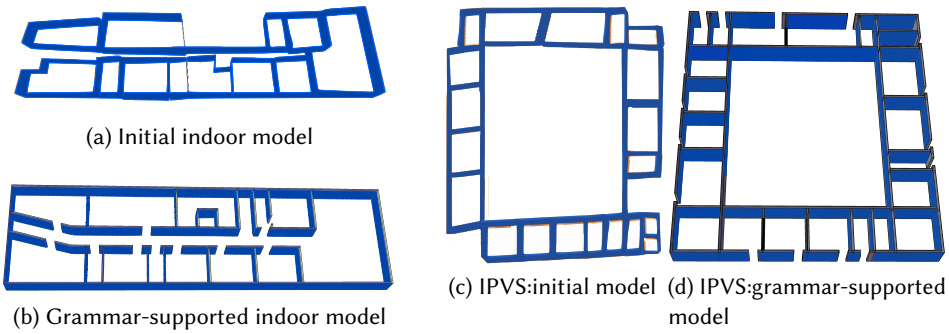


(a) Point cloud of a room (b) Planes detected using the RHT algorithm (c) Vertical and horizontal planes

Fig. 14. Results of plane surface detection

another processing step which employs the indoor grammar as a model refinement tool. To this end, we project the initial model onto the x-z plane, before estimating the distance between the walls perpendicular to the reference line segments (cf. Section 4.2). In Figure 15a, the RLS lines are those lines surrounding the corridor area. After extracting the rooms width, they are used as an input, i.e. observables, to the HMM model. Executing the Viterbi algorithm over the HMM

model results in a set of grammar rules. Through assigning this set of rules along the RLS line, we obtain a final room layout as shown in Figure 15b for the ifp floor and in Figure 15d for the IPVS institute. Obviously, the final indoor models are highly accurate than the initial models where the structural knowledge—encoded in the grammar—widely helps in improving the modeling accuracy (cf. Section 7.3).



(a) Initial indoor model

(b) Grammar-supported indoor model

(c) IPVS:initial model

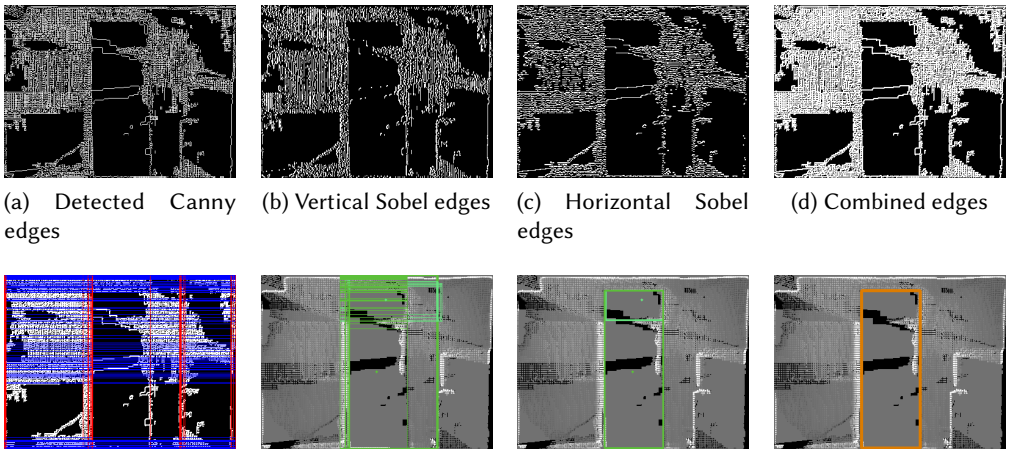
(d) IPVS:grammar-supported model

Fig. 15. Indoor models obtained by MapSense

For the doors detection, each depth image—generated using ray tracing—is used as an input to a combination of edge detection algorithms. Figure 16a demonstrates a wall surface whose edges have been detected using the Canny algorithm. As aforementioned, adopting solely the Canny algorithm is not recommended owing to missing many edges. Therefore, we also adopted the Sobel filter to detect the vertical and horizontal edges, as depicted in Figures 16b and 16c. To enhance the detection accuracy, we combined the Canny and Sobel edges (cf. Figure 16d) so that the Hough lines algorithm can properly detect the various lines in the depth images. Figure 16e demonstrates the vertical and horizontal lines detected through converting each point from the Cartesian space to the Hough space. Both of the vertical and horizontal lines are mostly concentrated in the area which surrounds the actual door object. Nevertheless, there exist also other misleading lines (cf. blue lines in the middle of Figure 16e) detected due to the noisy nature of the collected point clouds. As a result, a myriad of bounding boxes will be constructed through connecting any two vertical lines and any two horizontal lines.

Figure 16f depicts the set of bounding boxes which have been classified by the random forest algorithm to be potential door objects. Obviously, there exist several bounding boxes representing the same object (cf. the boxes at the top of Figure 16f). Figure 16g demonstrates the output of the k-Means clustering algorithm which grouped similar objects according to their area and center point. As can be seen in the figure, there exist only two clusters each comprises a single bounding box. Finally, we combine both bounding boxes to inscribe a new box (cf. orange box in Figure 16h) representing the actual door object. Such a doors detection process is repeated for each wall segment. Figure 15b shows the grammar-supported model where each detected door object is delineated as an opening in the wall surface.

For furniture detection, we leverage a random forest classifier to infer the furniture objects. In these experiments, we confined ourselves to the inference of table and chair objects. To differentiate between tables and chairs, we basically set a reasonable constraint that the Euclidean distance between horizontal segments—representing the table objects—and the floor segment has to be greater than 0.7m. Such a constraint is typically valid for tables and chairs existing in public buildings. As an example, Figure 17a depicts the top view of a point cloud where the blue area

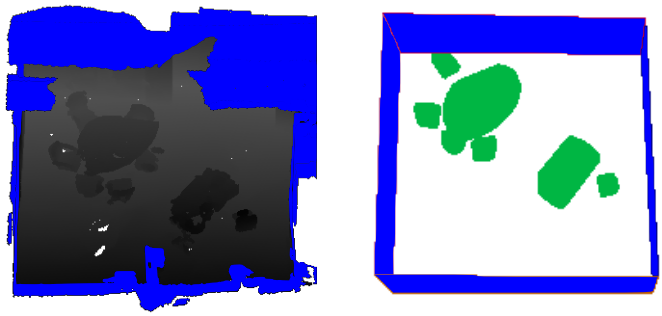


(a) Detected Canny edges (b) Vertical Sobel edges (c) Horizontal Sobel edges (d) Combined edges

(e) Detected Hough lines (f) Classified potential door objects (g) Final clusters (h) Final door object edges

Fig. 16. Various steps of doors detection

represents a ceiling and four walls. In such a room, there are five chairs and two tables (cf. black segments in the middle of Figure 17a). The random forest classifier—whose maximum number of trees is set to eight—labels each segment as either table or chair object, i.e. the number of categories equals two. Figure 17b depicts the reconstructed room after successfully recognizing the horizontal segments which represent either chair or table objects (cf. green segments in Figure 17b).

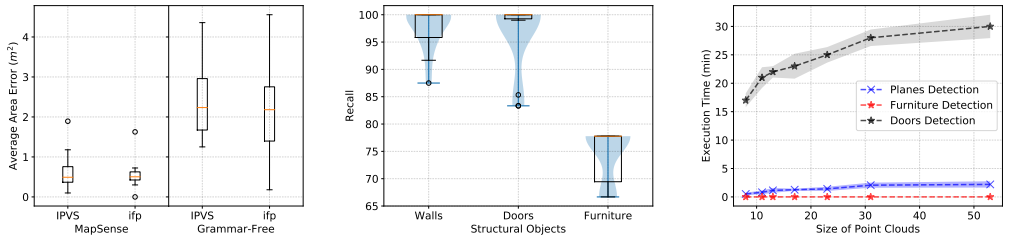


(a) Original point cloud of a room (b) Detected furniture objects

Fig. 17. Furniture classification using random forest

7.3 QoS Metrics

In this section, we estimate the performance of our proposed pipeline in terms of various QoS metrics. Specifically, we estimate the detection and modeling accuracy at the back-end server, before evaluating the energy overhead on the mobile devices. Each run of these measurements was repeated ten times and the resultant values are then averaged.



(a) Modeling accuracy (b) Detection accuracy (c) Porcessing time

Fig. 18. QoS results at the back-end server

7.3.1 Back-end Server. As a back-end server, we employed an Intel Core 2 machine running at 2.4 GHz and equipped with 8GB of RAM. Figure 18a shows a comparison of the modeling accuracy between the initial indoor model and the grammar-supported indoor model. Specifically, we estimated the average area error, i.e. defined as the difference between the actual and detected area of each room, for both generated models. As depicted in Figure 18a, MapSense (i.e. grammar-supported model) has much higher accuracy than the initial grammar-free model (on average less errors by 76% for the ifp dataset and by 74% for the IPVS dataset). The error distribution in case of the grammar-free model has relatively high variability, depending on the users' accuracy while collecting the point clouds in both datasets. In case of the grammar-supported model, the distribution is highly tight where the maximum error does not exceed circa 1.2 m^2 compared to circa 4.5 m^2 for the grammar-free model.

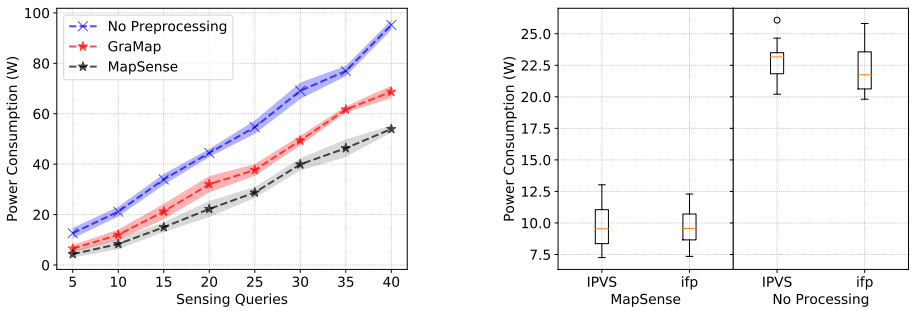
Figure 18b depicts the accuracy of detecting the various structural objects, including walls, doors, and furniture. To this end, we utilize the *recall* metric as a measure of the detection accuracy. The recall is defined as the fraction of relevant instances that are retrieved, i.e. $\frac{TP}{TP+FN}$ where TP and FN denote true positive and false negative, respectively. We ran the proposed pipeline over all the collected point clouds to estimate the true positives and false negatives. As shown in Figure 18b, our proposed pipeline achieves high detection accuracy, i.e. on average 97.6% for walls, 96% for doors, and 74.6% for furniture objects. Obviously, both walls and doors have high recall compared to the furniture objects. The reason is that some horizontal segments—representing furniture objects—are found to be merged together, i.e. two close chairs may be recognized as a single object. For the wall objects, the recall distribution is mostly concentrated in the range between 96% and 100%. Obviously, doors detection has relatively high variability since the highly-occluded areas of a given wall segment might be mistakenly annotated as "occupied". Nevertheless, the figure shows that the recall distribution of doors detection is also concentrated above 95%.

Figure 18c delineates the time elapsed while executing the various processing steps on a set of point clouds of different sizes. Specifically, we compare the execution time of planes detection, furniture detection, and doors detection. As shown in the figure, the elapsed time gradually increases with the size of the collected point clouds. Moreover, doors detection requires much more time than planes detection (at most by 94%) and than furniture detection (at most by 99%). The reason is that ray tracing typically examines every point in each wall segment. Moreover, extracting the various features—required for classification—consumes a considerable amount of time.

7.3.2 Mobile Devices. In this section, we evaluate the performance of MapSense in terms of the power consumption of the mobile devices due to participating in the crowdsensing activities. We measured the energy required for (1) capturing the point clouds, (2) preprocessing them, and (3)

reporting them to the crowdsensing server. We mainly evaluate the effectiveness of the probabilistic quality model, the voxel grid downsampling, and the Octree compression in reducing the energy overhead on the mobile devices. For the power measurements, we found that our Monsoon power monitor is unable to power the Tango devices. The latter requires a 7.5 Volts power supply whereas the Monsoon power monitor can only supply a maximum of 4.5 Volts. To measure the energy consumption of Tango devices, we recorded the battery's current and voltage every 100 ms while disabling WiFi, and no other background activity existed. The screen energy (at 50 % brightness level) has been subtracted from the total energy consumption to solely consider the energy overhead of running the algorithms. Each run of these measurements was repeated ten times and the resultant values are then averaged.

Figure 19a depicts a comparison of the accumulated power consumption between MapSense and a baseline method, referred to as GraMap [1]. Specifically, we measured the power consumption due to responding to several sensing queries ranging from five to 40 queries. For each query, we measured the power consumed for generating the point cloud, executing the processing pipeline on the mobile device, and reporting the processed point cloud to the back-end server. The mobile-side processing in GraMap is similar to that of MapSense except that downsampling is not involved in GraMap. For the Octree compression, we selected the low-quality profile since it requires a shorter time compared to the medium-quality and high-quality profiles. In addition to GraMap, we compare MapSense to the case of "No Processing" where the captured point clouds are directly reported to the crowdsensing server without any processing. As depicted in the figure, MapSense outweighs both baseline methods consuming, on average, 24% less energy than GraMap and 46% less energy than the "No Processing" method.



(a) Comparing MapSense relative the baseline methods (ifp dataset)

(b) Results of both datasets

Fig. 19. Accumulated power consumption on the mobile devices

Figure 19b provides a comparison between the accumulated power consumed while collecting several bundles of the mapping data in both use cases, i.e. the IPVS floor and the ifp floor. Nearly, the adopted methods consume a comparable amount of energy in both buildings where MapSense consumes less energy (at least by 57% for the IPVS floor and 55% for the ifp floor). To sum up, the evaluation results confirmed that MapSense can derive highly-accurate indoor models while reducing the energy costs of preprocessing and reporting the point clouds.

7.4 Discussion

As the evaluation results depict, our MapSense approach managed to derive highly-accurate indoor models thanks to exploiting our indoor grammar. However, our indoor grammar still suffers from being limited to buildings with regular structures, which typically can be found in office buildings, hospitals, the buildings on campus, etc. In other words, our indoor grammar is currently not able to handle curved and overlapped structures. For instance, it is clear from comparing the initial indoor model and the grammar-supported model of the IPVS dataset that the right bottom overlapped rooms were modeled as a single room (cf. Figure 15). To sidestep this drawback, another layer of split grammars have to be utilized in order to derive rules over overlapped objects. Aside from being limited to regular structures, our current indoor grammar is still not able to model 3D objects, such as opening and furniture objects. It is crucial to model such objects where they represent context information in many indoor LBSs. Currently, we develop a 3D indoor grammar for modeling workplaces with their various structural objects. Figure 20 demonstrates an example of the rules derivation process for modeling a workplace consisting of six workplaces, where the green boxes represent tables and the yellow boxes represent chairs. The iterative process starts from a free space, called face, which can be split in each iteration. In this case, we start with choosing a reference line by selecting the face's largest edge (i.e. preferably edges containing windows). Afterward, the split rules are applied according to their probabilities until no further rule can be placed on the reference line. The figure on the right is evolving in each iteration till reaching the figure on the left where no free spaces exist.

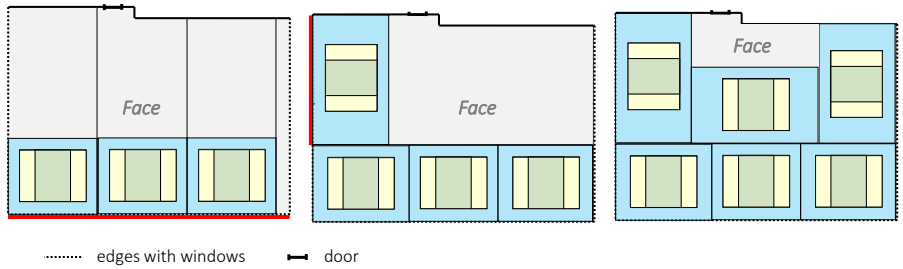


Fig. 20. 3D indoor grammar for modeling workplaces

Since MapSense relies on the concept of participatory sensing in which the users voluntarily participate in contributing information, our proposed system has to provide incentives for the users who traverse the indoor areas to participate in data collection. One of the most important incentives is reducing the energy overhead on their mobile devices while answering sensing queries. Therefore, MapSense employed several techniques to reduce such an overhead, including voxel-grid downsampling, Octree compression and the adoption of the probabilistic quality model. The latter approach proved its ability in reducing the power consumption while collecting point clouds by novice users. However, it is important to mention that the performance of such a quality model is highly affected by the scanned environment. For instance, if the area—to be scanned—comprises a large number of glass walls, our quality model becomes extremely effective where the sensing modalities of modern mobile devices are not able to collect depth information while scanning glass walls, thus leading to a large number of missing walls.

8 RELATED WORK

In this section, we discuss related work in the realm of indoor mapping and structural objects inference. In fact, the problem of automatically deriving indoor models has been tackled in several research works [11, 18, 24]. For instance, MapGENIE [24] derives an initial floor plan from a set of inaccurate crowdsensed motion traces. Afterward, the inaccurate traces are refined using information about the building's exterior. Along a similar line, iMap [18] detects the floor plan as well as higher-level objects, e.g. stairs, escalators, elevators and doors. To this end, iMap relies on collecting several types of data, including motion traces, atmospheric pressure, audio, and Wi-Fi signal strength. Obviously, these methods require the users to respond to several sensing queries to collect sufficient data generated by different sensors. Alternatively, MapSense requires solely one sensing query per area for reconstructing such an area.

Similarly, there exist several other efforts which focused on modeling the indoor environment using 3D point clouds [1–3, 15]. For example, we introduced GraMap [1] as a QoS-aware automatic indoor modeling method using crowd-sourced 3D point clouds. GraMap leverages region growing segmentation and indoor grammars to generate highly-accurate 2D indoor maps. Later, we introduced GreenMap [15], as an extension of GraMap, to further reduce the energy overhead through preprocessing the point clouds on the mobile devices for the sake of reporting much less number of 3D points. Specifically, GreenMap leverages the concept of approximate computing to enable energy-aware processing of the 3D point clouds on "resources-constrained" mobile devices. To further reduce the energy overhead on the mobile devices, we introduced ComNSense [2] as an energy-aware 3D data acquisition from mobile devices. ComNSense leverages the indoor grammar the mobile devices to detect the walls location, before reporting these small data to the crowdsensing servers for reconstructing the indoor maps. Nevertheless, these efforts are limited to the generation of 2D indoor maps.

As aforementioned, MapSense takes as an input a set of registered point clouds from modern mobile devices to recognize the various structural objects, including ceiling, floor, walls, and doors. It is worth mentioning that these registered point clouds could also be obtained using handheld Kinect or using mobile robots. In this realm, Labbé and Michaud [16] propose RTAB-Map, a graph-based simultaneous localization and mapping (SLAM) approach with loop closure detection. RTAB-Map can be used with a handheld Kinect, a stereo camera or a 3D lidar to generate 3D point clouds of the environment and to create a 2D occupancy grid map for navigation. RTAB-Map mainly relies on an efficient memory management mechanism to keep computation time for each new observation under a fixed time limit, thus respecting the real-time limit for long-term operation. In addition to these traditional methods, Gao et al. [11] propose Jigsaw, a technique for generating 3D point clouds from a large set of 2D images of the environment. To this end, the authors exploit two computer vision techniques, namely structure from motion and vanishing line detection.

To generated 3D models, Adan and Huber [3] utilized laser scanner data to model predominantly planar surfaces, such as walls, floors, and ceilings. They leverage 3D Hough transform and support vector machine (SVM) to detect the wall surfaces and the doors objects, respectively. As aforementioned, collecting point clouds using laser scanners is a time- and effort-consuming task. Moreover, the SVM algorithm might lack the required precision while classifying the possible door objects. Along a different line, MapSense exploits the concept of crowdsensing to collect the 3D data to generate detailed indoor models. Furthermore, random forest—adopted in MapSense—has been widely proven to be much more accurate than the SVM algorithm. One key distinction between MapSense and the state-of-the-art methods is that our method provides a comprehensive processing pipeline for the automatic indoor model generation while considering the various QoS metrics, such as the execution time, the modeling accuracy, and the energy overhead on the mobile device.

Aside from generating indoor maps, there exist other research activities which focused on inferring the high-level structural objects, i.e. doors, chairs, and tables [12, 17, 22]. In [22], Nikoohemat et al. presented a processing pipeline for the detection of openings, doors, and reflected surfaces. They leverage a combination of voxels and trajectory to detect open and closed doors intersected by the trajectory. However, their evaluations showed that precision of such an approach needs to be highly improved. Lau et al. [17] introduced a framework for classifying a certain 3D object into a specific object type, e.g. chair or table. To this end, they employed a formal grammar which encodes the main features of such 3D objects. However, the main limitation of this work is that the grammar and expert rules are defined manually. Alternatively, Günther et al. [12] proposed approach for recognizing furniture objects in point clouds based on structural descriptions from an OWL-DL ontology. Compared to these approaches, MapSense adopts a simpler machine learning-based method for inferring tables and chairs. Such a lightweight algorithm—as clarified in Section 7.3—enables its adoption in real-time applications, as clarified in Section 7.3.

9 CONCLUSION & FUTURE WORK

In this article, we tackled the problem of generating indoor models through inferring the structural objects, i.e. walls, ceiling, floor, doors, and furniture. To this end, we introduced MapSense, as an automatic indoor model generation approach using crowdsensed 3D point clouds. MapSense represents a multi-step process comprising wall surface detection, grammar-based refinement, doors detection, and furniture classification. The performance evaluation showed that indoor grammars highly improve the modeling accuracy. Moreover, doors detection using a combination of ray tracing and random forest has proven to be effective and accurate, despite requiring relatively long execution time. MapSense also reduced the energy overhead on the mobile devices thanks to the adopted preprocessing steps. A logical extension of this work involves developing a 3D grammar which can be used to directly model the 3D objects. Moreover, we plan to further improve the accuracy of our furniture detection method to avoid the problem of connected segments. We also aim at reducing the execution time of our doors detection method through approximating the ray tracing computations.

ACKNOWLEDGMENT

The authors would like to thank Lavinia Runceanu, for her help while collecting the mapping data.

This work is supported by the German Federal Ministry of Education and Research (BMBF) grant 01DH17059 and the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG), grants FR 823/25-1 and RO 1086/17-1.

REFERENCES

[1] M. Abdelaal, F. Dürr, K. Rothermel, S. Becker, and D. Fritsch. 2017. GraMap: QoS-Aware Indoor Mapping Through Crowd-Sensing Point Clouds with Grammar Support. In *Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous 2017)*. ACM, 292–302.

[2] M. Abdelaal, D. Reichelt, F. Dürr, K. Rothermel, L. Runceanu, S. Becker, and D. Fritsch. 2018. Comnsense: Grammar-driven crowd-sourcing of point clouds for automatic indoor mapping. *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2018), PACM IMWUT Issue 1* 2, 1 (2018), 1.

[3] A. Adan and D. Huber. 2011. 3D reconstruction of interior wall surfaces under occlusion and clutter. In *2011 International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission*. IEEE, 275–281.

[4] N. Akkiraju, H. Edelsbrunner, M. Facello, P. Fu, E. Mucke, and C. Varela. 1995. Alpha shapes: definition and software. In *Proceedings of the 1st International Computational Geometry Software Workshop*, Vol. 63. 66.

[5] ATAP. 2017. Google Project Tango. https://developers.google.com/project-tango/ accessed on June 2016.

[6] S. Becker, M. Peter, D. Fritsch, D. Philipp, P. Baier, and C. Dibak. 2013. Combined Grammar for the Modeling of Building Interiors. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* II-4/W1 (2013), 1–6.

[7] G. Bertoline, E. Wiebe, C. Miller, and L. Nasman. 2005. *Fundamentals of graphics communication*. Ray Tracing Chapter.

[8] J. Blanco. 2010. *A Tutorial on SE(3) Transformation Parameterizations and On-Manifold Optimization.* Technical Report 3. University of Malaga.
[9] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (01 Oct 2001), 5–32.
[10] G. Forney. 1973. The Viterbi Algorithm. *Proc. of the IEEE* 61, 3 (1973), 268–278.
[11] R. Gao, M. Zhao, T. Ye, F. Ye, G. Luo, Y. Wang, K. Bian, T. Wang, and X. Li. 2016. Multi-story indoor floor plan reconstruction via mobile crowdsensing. *IEEE Transactions on Mobile Computing* 15, 6 (2016), 1427–1442.
[12] M. Günther, T. Wiemann, S. Albrecht, and J. Hertzberg. 2017. Model-based Furniture Recognition for Building Semantic Object Maps. *Artificial Intelligence* 247 (2017), 336–351.
[13] K. Habgood and I. Arel. 2012. A Condensation-based Application of Cramer's Rule for Solving Large-scale Linear Systems. *Journal of Discrete Algorithms* 10 (2012), 98–109.
[14] I. Jolliffe. 2002. *Principal Component Analysis.* Wiley Online Library.
[15] J. Kässinger, M. Abdelaal, F. Dürr, and K. Rothermel. 2018. GreenMap: Approximated Filtering Towards Energy-Aware Crowdsensing for Indoor Mapping. In *2018 IEEE 15th International Conference on Mobile Ad Hoc and Sensor Systems (MASS).* IEEE, 451–459.
[16] M. Labbé and F. Michaud. 2019. RTAB-Map as an Open-source LIDAR and Visual Simultaneous Localization and Mapping Library for Large-scale and Long-term Online Operation. *Journal of Field Robotics* 36, 2 (2019), 416–446.
[17] M. Lau, A. Ohgawara, J. Mitani, and T. Igarashi. 2011. Converting 3D Furniture Models to Fabricatable Parts and Connectors. In *ACM SIGGRAPH 2011 Papers (SIGGRAPH '11).* ACM, New York, NY, USA, Article 85, 6 pages. https://doi.org/10.1145/1964921.1964980
[18] C. Luo, H. Hong, L. Cheng, K. Sankaran, and M. Chan. 2015. iMap: Automatic Inference of Indoor Semantics Exploiting Opportunistic Smartphone Sensing. In *Sensing, Communication, and Networking (SECON), 2015 12th Annual IEEE International Conference on.* IEEE, 489–497.
[19] G. I. Maps. [n. d.]. Google Indoor Maps Availability. https://www.google.com/maps/about/partners/indoormaps/..
[20] B. Marques, R. Carvalho, P. Dias, M. Oliveira, C. Ferreira, and B. Sousa Santos. 2018. Evaluating and Enhancing Google Tango Localization in Indoor Environments using Fiducial Markers. In *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC).* IEEE, 142–147.
[21] J. Monson, M. Wirthlin, and B. Hutchings. 2013. Optimization Techniques for a High Level Synthesis Implementation of the Sobel Filter. In *2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig).* IEEE, 1–6.
[22] S Nikoohemat, M Peter, S Oude Elberink, and G Vosselman. 2017. Exploiting Indoor Mobile Laser Scanner Trajectories for Semantic Interpretation of Point Clouds. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences* 4 (2017).
[23] A. Nüchter and K Lingemann. 2011. 3DTK–The 3D Toolkit.
[24] D. Philipp, P. Baier, C. Dibak, F. Durr, K. Rothermel, S. Becker, M. Peter, and D. Fritsch. 2014. Mapgenie: Grammar-enhanced Indoor Map Construction from Crowd-sourced Data. In *2014 IEEE International Conference on Pervasive Computing and Communications (PerCom).* IEEE, 139–147.
[25] L. Rabiner. 1989. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proc. of the IEEE* 77, 2 (Feb 1989), 257–286.
[26] R. Roberto, J. Lima, T. Araújo, and V. Teichrieb. 2016. Evaluation of Motion Tracking and Depth Sensing Accuracy of the Tango Tablet. In *Mixed and Augmented Reality (ISMAR-Adjunct), 2016 IEEE International Symposium on.* IEEE, 231–234.
[27] L. Runceanu, S. Becker, N. Haala, and D. Fritsch. 2017. Indoor Point Cloud Segmentation for Automatic Object Interpretation. *Proceedings of the 37. Wissenschaftlich-Technische Jahrestagung der DGPF* (2017).
[28] R. Rusu and S Cousins. 2011. Point Cloud Library (pcl). In *2011 IEEE International Conference on Robotics and Automation.* 1–4.
[29] J. Snyder and A. Barr. 1987. *Ray Tracing Complex Models Containing Surface Tessellations.* Vol. 21. ACM.
[30] J. Talton, Y. Lou, S. Lesser, J. Duke, R. Měch, and V. Koltun. 2011. Metropolis Procedural Modeling. *ACM Trans. Graph.* 30, 2, Article 11 (April 2011), 14 pages.
[31] M. Weinmann. [n. d.]. *Preliminaries of 3D Point Cloud Processing.* Springer Int. Publishing, Cham, 17–38.
[32] Mathworld Wolfram. [n. d.]. PLane-Plane intersection. http://mathworld.wolfram.com/Plane-PlaneIntersection.html. Retrieved on November 2018.
[33] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky. 2003. *Instant architecture.* Vol. 22. ACM.
[34] L. Xu, E. Oja, and P. Kultanen. 1990. A New Curve Detection Method: Randomized Hough Transform (RHT). *Pattern recognition letters* 11, 5 (1990), 331–338.