GreenMap: Approximated Filtering towards Energy-Aware Crowdsensing for Indoor Mapping

Johannes Kässinger, Mohamed Abdelaal, Frank Dürr, Kurt Rothermel
Institute of Parallel and Distributed Systems, University of Stuttgart
Email: first.last@ipvs.uni-stuttgart.de

***Abstract*—Recently, mobile crowdsensing has become an appealing paradigm thanks to the ubiquitous presence of powerful mobile devices. Indoor mapping, as an example of crowdsensing-driven applications, is essential to provide many indoor location-based services, such as emergency response, security, and tracking/navigation in large buildings. In this realm, 3D point clouds stand as an optimal data type which can be crowdsensed—using currently-available mobile devices, e.g. Google Tango, Microsoft Hololens and Apple ARKit—to generate floor plans with different levels of detail, i.e. 2D and 3D mapping. However, collecting such bulky data from "resources-limited" mobile devices can significantly harm their energy efficiency. To overcome this challenge, we introduce GreenMap, an energy-aware architectural framework for automatically mapping the interior spaces using crowdsensed point clouds with the support of structural information encoded in formal grammars. GreenMap reduces the energy overhead through projecting the point clouds to several filtration steps on the mobile devices. In this context, GreenMap leverages the potential of *approximate computing* to reduce the computational cost of data filtering while maintaining a satisfactory level of modeling accuracy. To this end, we propose two approximation strategies, namely DyPR and SuFFUSION. To demonstrate the effectiveness of GreenMap, we implemented a crowdsensing Android App to collect 3D point clouds from two different buildings. We show that GreenMap achieves significant energy savings of up to 67.8%, compared to the baseline methods, while generating comparable floor plans.**

I. Introduction

Nowadays, location-based services enjoy ever-growing popularity thanks to their significant role in supporting various Internet of things and mobile computing applications. In this context, indoor modeling became an indispensable necessity for providing indoor location-based services, e.g. emergency response, or indoor navigation/routing in airports, universities, hotels, and other public buildings. Collecting mapping data using traditional manual techniques is an extremely time-consuming and cumbersome task. Hence, several researchers have harnessed the potential of crowdsensing [1] for automatically mapping the interior spaces [2]–[6]. In this realm, we introduced an automatic modeling approach for 2D floor plans that relies on Markov chains to integrate a set of crowdsensed motion traces and a formal indoor grammar [2]. Afterwards, we proposed the crowdsensing of 3D point clouds for 3D indoor modeling applications [5]. Literally, point clouds are 3D data structures representing points in space where the points usually represent the X, Y, and Z geometric coordinates of a sampled surface. In fact, point clouds can broadly overcome shortcomings of other data types, e.g. motion traces, images, and WiFi footprints. These traditional data types are often limited to the derivation of 2D maps together with requiring large data sets to derive the mapping process, e.g. Jigsaw [4] collects about 150 images for modelling a single landmark. Alternatively, a small number of point clouds is typically sufficient to generate a floor plan with several levels of detail (i.e. 2D and 3D). For instance, one point cloud per room was enough to derive an indoor model in [5].

Aside from being well-suited for indoor modeling, point clouds are naturally bulky data to be processed and uploaded by mobile devices to a crowdsensing server. In [5], the participating mobile devices have to sample and report several point clouds whose size rang from five MBytes to 50 MBytes. Despite adopting Octree compression [7], the energy costs of performing compression and reporting the results are relatively high (cf. evaluations in Section VI). Thus, improving energy efficiency is crucial for mobile users' acceptance to participate in crowdsensing indoor models. Accordingly, a challenge of reducing this energy overhead while preserving the quality of the generated indoor models emerges. To tackle this challenge, this paper introduces the GreenMap approach for the automatic derivation of 2D floor plans while increasing the energy efficiency of crowdsensing for indoor modelling.

The basic idea of GreenMap is to reduce the energy overhead of acquiring point clouds through filtering out irrelevant points, e.g. furniture and noise. To minimize the processing overhead of these filters, GreenMap exploits the potential of *approximate computing* to reduce the amount of computations at the expense of a tolerable loss in the accuracy of the results. In general, approximate computing is a wide spectrum of techniques that relaxes the accuracy of computation in order to improve other performance metrics, e.g. energy efficiency and latency. Practically speaking, approximate computing can be implemented through: (1) reducing the precision of approximable parts via parameter tuning or substitution [8]; (2) skipping tasks, memory accesses, or some iterations of a loop (aka loop perforation) [9]; and/or (3) performing an operation on inexact hardware [10].

In this context, we propose two different approximation strategies, namely DyPR and SuFFUSION. The former strategy relies on the concept of loop perforation, employing a probabilistic model to skip the processing of a certain set of data points. The latter strategy relies on substitution of energy-intense elements and function fusion, where it smartly integrates the down-sampling and the normals-based filters to

 Published in proceedings of the 2018 IEEE 15th International Conference on Mobile Ad-hoc and Sensor Systems (MASS2018), Chengdu, China 2018. http://dx.doi.org/10.1109/MASS.2018.00069

skip unnecessary computations. To the best of our knowledge, GreenMap represents the first work that exploits approximate computing to improve the energy efficiency in crowdsensing-driven applications for 2D/3D indoor modeling.

In detail, the paper makes the following contributions: (1) We define an architectural framework—built upon a previous work by our group [5]—for modeling indoor environments using crowdsensed 3D point clouds. Our framework first generates an initial model through processing the collected point clouds. Subsequently, we exploit the design-time knowledge, i.e. typical structural information of buildings encoded in a formal grammar to refine the initial model to improve the modeling accuracy. (2) We devise an energy efficient processing pipeline on the mobile device to report only the relevant parts of the point clouds. To this end, we employ two filters, namely *voxel grid* and *normals-based filtering*. The former filter down-samples a point cloud through restructuring it onto a homogeneous grid before estimating the centroid in each cell. Since we are interested only in the wall segments, we use the eigenvectors obtained by the principal component analysis (PCA) [11], i.e. orientation of each point, to filter out the points which do not belong to the wall segments. (3) We introduce two different approximation strategies, namely DyPR and SuFFUSION, to significantly reduce the computational cost of data processing on the mobile devices via identifying parts of the computation that can be approximated without violating the quality of the results. (4) We present a proof-of-concept implementation and evaluation of GreenMap in a real-world scenario. We implemented an Android App to collect more than 135 point clouds (circa 3 GBytes of collected data). To overcome the localization errors, the Android App integrates the main three features of Tango technology, namely *motion tracking*, *depth perception*, and *area learning* [12]. The results show that using GreenMap, we obtain a significant improvement of the energy efficiency up to a reduction of 67.8% relative to the baseline approaches with a comparable modeling accuracy.

The remainder of this paper is organized as follows: Section II introduces the system model with describing the main parts of our processing pipeline (cf. Figure 1). In Section III, we describe the processing tasks implemented on the mobile devices to filter out and to down-sample the point clouds. Before delving into the server-side processing, Section IV introduces our approximation strategies which have been adopted to reduce the energy burden on the participating mobile devices. Section V discusses the processing steps required to generate the final model, including the formal grammar integration with the point clouds. Section VI presents our real-world performance evaluations in terms of the modeling accuracy and the energy consumption of the mobile devices. Finally, Section VII discusses recent work in the realm of indoor mapping and crowdsensing before Section VIII concludes the paper with an outlook on future work.

II. System Overview

Our system encompasses two primary components: (1) a set of participating mobile devices $\mathcal{D} = d_1, \cdots, d_n$ capable of collecting 3D point clouds $\mathcal{P} = P_{(1,1)}, \cdots, P_{(m,n)}$ where $P_{(m,n)}$ is the point cloud generated in response to the sensing query q_m by the mobile device d_n; (2) a set of back-end servers, which send the sensing queries and receive the point clouds $\mathcal{P}$ from the mobile devices $\mathcal{D}$. A sensing query q in indoor modeling applications typically comprises the location of the area to be scanned and the minimum quality requirements. In this paper, we assume a single back-end server communicating with the mobile devices via a WiFi network. The system consists of three sequential processing blocks, namely *the point cloud acquisition*, *the initial model generation* and *the grammar-based indoor modeling* (cf. Figure 1). The mobile device receives sensing queries from the back-end server to scan specific areas. In this regard, we assume that the mobile devices $\mathcal{D}$ are aware of their location in the building [6]. To save the energy consumption of the mobile devices, two processing tasks, i.e. down-sampling and normals-based filtering, are to be implemented prior to the Octree compression. Once received by the back-end server, the point clouds are processed by the *initial modeling* block (cf. Figure 1) to generate a model which still suffers from inaccuracies. In the *grammar-based modeling* block, we exploit the floor plans of other buildings to generate a formal grammar which is then used to refine the initial model. The concept of an indoor grammar provides a powerful mean to encode structural information for different kind of architectural domains. The initial indoor model can be highly improved through exploiting the grammar's embedded information, e.g. the dimensions of rooms, the number of rooms, the relative room ordering, geometric constraints, etc. To this end, we design a Hidden Markov model (HMM) to integrate the formal grammar with the generated initial model. The final output of our system is a highly-accurate 2D indoor map of the scanned areas.

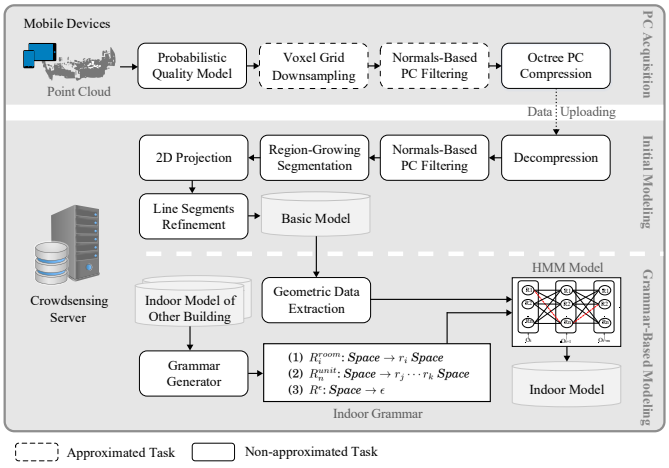


Fig. 1: GreenMap processing pipeline

In general, point clouds are relatively bulky data with thousands or even millions of 3D point, according to the size of the scanned spaces. Once a sensing query q is received by

a mobile device that is located in the sensing area, the depth sensor generates a point cloud p_{mem} which contains N_{mem} points. If the point cloud p_{mem} is directly compressed, its size will be reduced to N_{oct} points at the expense of consuming E_{oct} joule per point for performing compression. Despite tuning the Octree compression, the communication energy consumption is still relatively high compared with the energy required to collect other data types, such as motion traces and 2D images (cf. [6]). Knowing that point clouds are mostly redundant, we utilize a voxel grid filter to down-sample them so that a subset of the points—which is still representative of the original point cloud—can be further processed. Aside from redundancy, we target the generation of 2D floor plans. Hence, GreenMap uses a normals-based filter to avoid processing and reporting the points which do not belong to the wall segments.

Adopting these two filters drastically reduces the size of the point cloud p_{mem}, before the Octree compression that eventually generates a lightweight point cloud $\widehat{p}_{(m,n)}$. Nevertheless, these filters incur relatively heavy computations, thus the energy gain for communication obtained by reducing the number of points can be simply compensated. To minimize the computation costs of these two filters, we propose two independent approximation strategies to trade-off the energy cost of processing and the accuracy of the results. Accordingly, our objective function can be expressed by

$$\begin{aligned} &\text{minimize} \quad \underbrace{(N_{mem} \times E_{vg})}_{\text{voxel grid}} + \underbrace{(N_{vg} \times E_{pca})}_{\text{PCA-based filter}}, \\ &\text{subject to} \quad \Omega(p_{(m,n)}) - \Omega(\widehat{p}_{(m,n)}) \leq \delta. \end{aligned} \tag{1}$$

Equation 1 clearly expresses our objective in reducing the computational energy costs of the filters where E_{vg}, N_{vg} and E_{pca} denote the voxel grid's energy cost per point, the dimension of the down-sampled point cloud $p_{vg} \subset p_{mem}$ and the normals-based filter's energy cost per point, respectively. This objective is strictly governed by a quality constraint which dictates that difference between the modeling accuracy before the approximation $\Omega(p_{(m,n)})$ and after it $\Omega(\widehat{p}_{(m,n)})$ must not exceed a tolerable margin δ. The modelling accuracy Ω is defined in terms of the number of detected walls. In the next section, we provide an overview of the online processing pipeline, before explaining our approximation strategies in Section IV.

III. Online Processing Pipeline

In this section, we describe the processing tasks implemented on the mobile devices including the *voxel grid filter* and *PCA-based normal estimation*. Since point clouds are to be collected by non-experts, the depth data may suffer from low quality, i.e. being incomplete, inaccurate, and/or oblique. To avoid repeating the sensing queries due to reporting low-quality point clouds, GreenMap adopts a quality checkpoint on the mobile devices. Specifically, we define an acceptance probability as a conditional probability of the depth sensor to generate a high quality point cloud. Afterwards, we adopt two filters to reduce the data reported to the server. Both filters are used to reduce the amount of redundant and irrelevant points within a point cloud. Below, we discuss the idea behind these two filters while highlighting their role in our online processing pipeline.

A. Voxel Grid Filter

Generally, the depth sensors generates point clouds which include spatial redundancy, i.e. existence of many points which give no further information about the object's shape. In this context, the *voxel grid filter* [13] reduces the amount of points m within a point cloud $P_{mem} \in \mathbb{R}^m$ through redundancy elimination. The basic idea is to subdivide the cloud into a set of identical voxels (3D boxes) $V = \{v_1, ..., v_i, ..., v_r\}$ with a resolution r where $r \ll m$, $P_{vg} \in \mathbb{R}^r$. The resolution r depends on the defined edge length ε of the boxes. At the outset, all points are sorted to have neighboring points next to each other and layer on layer bottom up. Subsequently, all points that belong to a voxel v_i, are then represented by one single point approximated by their centroid $c_i = \frac{\sum_{j=1}^{\rho} p_j}{\rho}$ where ρ is the number of points in v_i. Figure 2a shows an example of a voxel grid where the small grey dots represent the points $p_i \in P_{mem}$. The big red dots denote their representing centroids $c_i \in P_{mem}$ and hence the points $p_i \in P_{vg}$. Despite reducing the number of points in the recorded point clouds, the voxel grid filter still keeps a considerable amount of irrelevant points. Therefore, we adopt a second stage of filtration, referred to as normals-based filtering, through exploiting design-time knowledge.

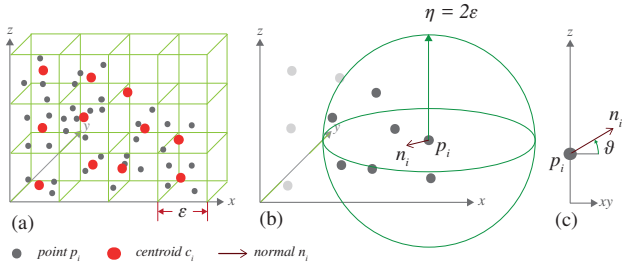


Fig. 2: Visualizing the idea behind our online filters: (a) point cloud P_{mem} aligned to a voxel grid, (b) nearest neighbour search for normal estimation, and (c) estimated normal's angle ϑ of $p_i \in P_{vg}$ towards the ground plane

B. Normals-Based Filter

To further reduce the amount of points in P_{vg}, we exploit the design-time knowledge which dictates that GreenMap currently targets the 2D modeling of public buildings. Hence, the crux behind our normals-based filter is to remove the irrelevant points belonging to non-wall segments, e.g. floor, ceiling or furniture. To this end. we estimate the normal n_i of each point $p_i \in P_{vg}$ before filtering out the points describing the non-wall segments. Literally, a normal is a vector orthogonal to a given object. For a point that belongs to a wall, its normal is ideally parallel to the ground plane and so $\vartheta = 0$ where ϑ is the angle towards the xy-plane. As depicted in Figure 2b, the normal n_i at point p_i is determined

by considering the surrounding points within a radius $\eta \leftarrow 2\varepsilon$, called *nearest neighbors*. By choosing a radius $\eta < 2\varepsilon$, the amount of found neighbors would not be sufficient as ε is the average distance between the points after applying the voxel grid filter. The surface normal is then computed using the *principal component analysis* (PCA) method [11].

For each point $p_i \in P_{vg}$, the PCA method estimates three eigenvector/eigenvalue pairs, where the eigenvector $\vec{v}$ represents a direction and the corresponding eigenvalue λ denotes the variance in this direction. Therefore, a covariance matrix Σ for a set of points within the defined radius η is estimated for all points $\in P_{vg}$ as described in Equation 2 [13], where $\overline{p}$ is the centroid. The eigenvectors are determined by solving $det(\Sigma - \lambda \cdot I) = 0$ where det is the determinant operator and I is the identity matrix. By solving Equation 3 with the given covariance matrix Σ and eigenvalues λ, the eigenvectors are determined. Owing to inaccuracies while recording the point clouds, the wall segments are often not perfectly parallel to the ground plane. Hence, we consider a safety margin θ_{th} while classifying the points to be part of a wall. Accordingly, the points whose normal angle is larger than the safety margin, i.e. $\vartheta > \theta_{th}$, are filtered out (cf. Figure 2c).

$$\Sigma = \frac{1}{k}\sum_{i=1}^{k}(p_i - \overline{p}) \cdot (p_i - \overline{p})^T \tag{2}$$

$$\Sigma \cdot \vec{v_j} = \lambda_j \cdot \vec{v_j}, j \in \{1, 2, 3\} \tag{3}$$

Despite being extremely effective in removing redundancy and irrelevant points, both filters can still add an energy overhead due to their computations (cf. evaluations in Section VI). To minimize such an overhead, we introduce in the following section two independent approximation strategies.

IV. Approximate Filtering

In this section, we introduce two different approximation strategies to approximate the computations of our online filters on the mobile devices.

A. DyPR – Dynamic Probabilistic Rounding

In this section, we introduce DyPR, a dynamic probabilistic rounding strategy inspired by Markov Chain Monte Carlo (MCMC) [14] for approximating the normals-based filter. Specifically, DyPR utilizes a probabilistic model to skip a few iterations while estimating the normals. Figure 3 describes the DyPR algorithm where the point cloud P_{vg} is used as an input to DyPR. We introduce a probability ω of the next point to be part of a wall. The algorithm begins with checking the status of a point $p_i \in P_{vg}$, i.e. whether it belongs to a wall (line 5). After detecting a point to belong to a wall, the algorithm performs probabilistic rounding to determine whether the next point p_{i+1} is also part of this wall (lines 7-9).

The probabilistic rounding is achieved by changing the probability ω in a logarithmic manner, i.e. $\omega_i = \frac{(2 \times \omega) + X_i}{3}$, where X_i is a binary variable whose value is set to one, i.e. increasing the probability, if p_i belongs to a wall and zero otherwise, i.e. decreasing the probability. To prevent the prediction part of ending up in an infinite loop, the probability ω is deliberately bounded in the range $[0.1, 0.9]$. To make a decision about the neighboring point, the probability ω_i is compared to a randomly-generated number according to the MCMC algorithm (lines 10,18). Despite achieving remarkable savings in the energy overhead, DyPR does not consider the voxel grid filter. Therefore, we investigated another approach which integrates the function of both filters while approximating their computations.

Require: point cloud P_{vg}
Initialization: point cloud $P_{DPR} \leftarrow \emptyset$ and $\omega \leftarrow 0.5$
1: **for all** $p_i \in P_{vg}$ **do**
2: $nn \leftarrow getNearestNeighbors(p_i, \eta)$
3: $n_i \leftarrow getNormalOfPoint(p_i, nn)$
4: $\vartheta_i \leftarrow getAngleTowardsXY(n_i)$
5: **if** $\vartheta_i \leq \theta_{th}$ **then** ▷ p_i is detected as part of a wall
6: $P_{DPR} \leftarrow P_{DPR} \cup p_i$
7: $increaseProbability(\omega)$ ▷ to max. 0.9
8: $p_i \leftarrow p_{i+1}$ ▷ move to the next point
9: $\omega_i \leftarrow \omega$
10: **while** $getRand(0, 1) \leq \omega_i$ **do**
11: $P_{DPR} \leftarrow P_{DPR} \cup p_i$
12: $\omega_i \leftarrow \omega_i^2$ ▷ probability propagation
13: $p_i \leftarrow p_{i+1}$ ▷ move to the next point
14: **else** ▷ p_i is detected as no part of a wall
15: $decreaseProbability(\omega)$ ▷ to min. 0.1
16: $p_i \leftarrow p_{i+1}$ ▷ move to the next point
17: $\omega_i \leftarrow (1 - \omega)$
18: **while** $getRand(0, 1) \leq \omega_i$ **do**
19: $\omega_i \leftarrow \omega_i^2$
20: $p_i \leftarrow p_{i+1}$ ▷ move to the next point
21: **return** P_{DPR}

Fig. 3: DyPR approximation strategy

B. SuFFUSION – Substitution and Function Fusion

The core idea behind SuFFUSION is to integrate the computations of both filters to skip the computational-intensive operations. From our experiments, we identified that the nearest neighbor search operation—executed for normals estimation within a radius $\eta \leftarrow 2\varepsilon$ of the downsampled point cloud P_{vg}—consumes on average 86% of the excessive time relative to

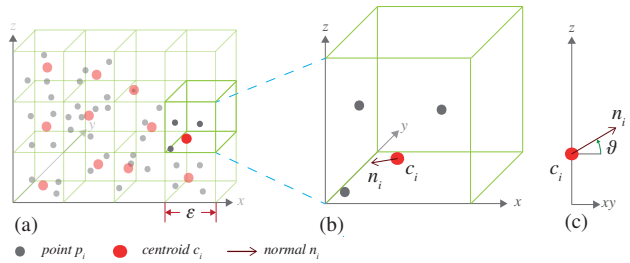


Fig. 4: Visualizing the SuFFUSION strategy: (a) Point cloud P_{mem} aligned to a voxel grid, (b) normal estimation while computing the centroid, and (c) normal's angle ϑ of c_i towards the ground plane

the other operations. As described in Section III-A, the basic function of the voxel grid filter is to divide the recorded point cloud P_{mem} into voxels and then represent each voxel using its centroid. Accordingly, we have to estimate the centroid through considering all points in each voxel. In this context, SuFFUSION entirely skips the nearest neighbor search operation and instead considers the same points that are used for the centroid calculation (cf. Figure 4b). Figure 5 describes the SuFFUSION algorithm where the input is the recorded point cloud P_{mem}. SuFFUSION utilizes the voxel $v_i \in V$ as input to normal estimation (line 4) while DyPR utilizes the nearest neighbor nn.

In this way, the radius for normals estimation is reduced from 2ε to only about 0.5ε. Since the walls are typically made of several not always perfectly overlapping segments, reducing the radius makes the filter more prone to inaccuracies while estimating the normals (cf. evaluations in Section VI-B). As a workaround, we decided to dynamically change the angle threshold as shown in Figure 4c based on the status of each point, i.e. whether it belongs to a wall. To this end, we introduce a tolerance factor f as an adaptive weight to the angle threshold θ_{th} (line 6). The tolerance factor f—bounded in the range $[0.5, 2]$—is continuously changed in a logarithmic manner where $f_{i+1} = \frac{f_i + Y_i}{2}$. The term Y_i denotes a binary variable whose value is set to one whenever the point p_i belongs to a wall and is set to zero otherwise (lines 8, 10). This adaptive threshold broadly makes SuFFUSION less vulnerable to not perfectly overlapped segments. The subsequent operations of the filter remain the same where points with $\vartheta \leq \theta th \cdot f$ are filtered out.

Require: point cloud P_{mem}, radius ε
Initialization: point cloud $P_{SuF} \leftarrow \emptyset$
1: $V \leftarrow devideIntoVoxels(P_{mem}, \varepsilon)$
2: **for all** $v_i \in V$ **do**
3: $c_i \leftarrow getCentroid(v_i)$
4: $n_i \leftarrow getNormalOfPoint(c_i, v_i)$ ▷ v_i instead of nn
5: $\vartheta_i \leftarrow getAngleTowardsXY(n_i)$
6: **if** $\vartheta_i \leq \theta_{th} \cdot f$ **then** ▷ c_i is detected as part of a wall
7: $P_{SuF} \leftarrow P_{SuF} \cup c_i$
8: $increaseToleranceFactor(f)$ ▷ to max 2
9: **else**
10: $decreaseToleranceFactor(f)$ ▷ to min 0.5
11: **return** P_{SuF}

Fig. 5: SuFFUSION approximation strategy

After explaining the online processing pipeline and our proposed approximation strategies, the next section explains the server-side processing pipeline developed to generate precise indoor models.

V. Server-Side Processing

At the back-end server, the point clouds collected from different mobile users are decompressed before being divided into a set of planar segments using the region growing segmentation [15]. The crux behind segmentation is to filter out any existent clutter, i.e. segments with non-planar surface representing furniture or noise. In fact, we ran the region-growing algorithm to divide a point cloud generated through scanning a medium-sized room into a bundle of planar surfaces on our mobile device. The mobile device took circa seven hours to cluster the point cloud. Therefore, we selected to segment the point clouds on the server. Subsequently, GreenMap projects the segments onto the x/y-plane resulting in several disjoint lines representing each wall. This fact occurs since some walls which separate neighboring rooms are often scanned twice from both sides. Accordingly, we refine these disjointed lines to eventually obtain an initial indoor model [5].

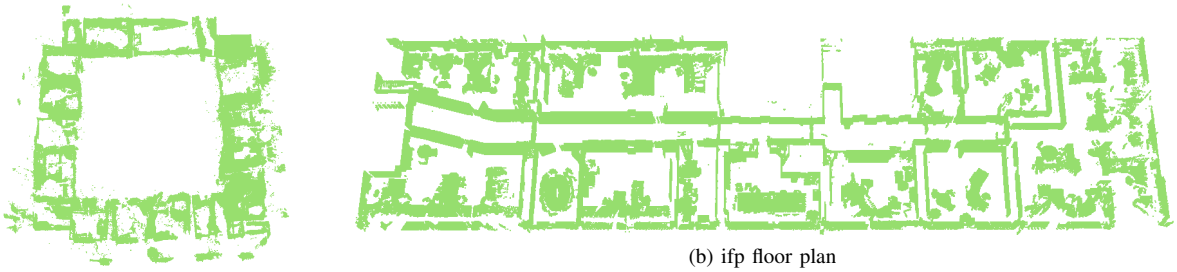
Since it suffers from inaccuracies and incompleteness, the initial model is then used as an input to the *grammar-enhanced modeling* component. In this component, we exploit the structural information encoded in formal grammars to generate a highly-accurate floor plan. To this end, we employ an indoor grammar which can be generated using the floor plan of other similar buildings. In [16], we provide an overview on how to generate the indoor grammar in more detail. Our indoor grammar comprise a set of terminal and non-terminal symbols which represent the rooms and hallways. Furthermore, it also encompasses a set of derivation rules for splitting the non-hallway areas into room sequence. To derive the final indoor model, GreenMap extracts the room size from the initial model to derive a hidden Markov model that generates the room layout. In this case, the hidden state represents the rooms located along the hallway areas. The room size—extracted from the initial model—is then utilized to adjust the emission probabilities of the different states. More details about the grammar-enhanced modeling component can be found in our previous paper [5].

VI. Evaluation

To demonstrate the effectiveness of our GreenMap approach, we implemented and tested the entire processing pipeline in a real-world scenario. We first describe the setup of our evaluations, before discussing the results for DyPR and SuFFUSION approximation methods using 3D data crowdsensed from two different buildings.

A. Experimental Setup

To sample point clouds from mobile devices, we utilized Tango mobile devices which are equipped with a 2.3 GHz quad-core CPU, 4 GB of RAM, 128 GB of internal flash storage, a 120° front-facing camera, a 4 MP RGB-IR rear-facing camera, and a 170° motion tracking camera [17]. To generate the point clouds, the mobile users have to scan the queried spaces using their Tango devices. To this end, we developed an Android App which integrates several features, such as depth perception and area learning, to generate point clouds in the form of depth information localized relative to a unique reference point. We integrated the *point cloud library* (PCL) [13] library into our software to process the generated point clouds. At the mobile devices, we implemented voxel



(a) IPVS floor plan

(b) ifp floor plan

Fig. 6: Combined point clouds

grid downsampling, normals estimation and Octree compression. Additionally, we implemented our proposed approximated filters, i.e. DyPR and SuFFUSION, before the Octree compression on the mobile devices (cf. Section II). All other components of the processing pipeline are implemented as a backend-service on a Linux server. Table I summarizes the parameters and their values which have been used throughout the evaluations.

TABLE I: Evaluation parameters

	no approx.	DyPR	SuFFUSION
edge length ε	4	4	4
radius η (cm)	8	8	(4)
angle θ_{th}	30	30	15-60

To test our implementation, we collected data from two floors from two different buildings at University of Stuttgart. The intuition is to examine the performance of GreenMap for mapping the indoor spaces in structurally-different environments (cf. Figure 6). During our evaluations, we collected several point clouds for each area within two days. Specifically, we generated 135 point clouds (i.e. approximately 3 GByte) from four volunteers in two floors belonging to two different institute buildings (i.e. *IPVS* and *ifp*). The volunteers were asked to freely move between the hallways and the different rooms before reporting their data to the crowdsensing server. Below, we first start with evaluating the impact of our two approximation strategies implemented in the voxel grid and the normals-based filter. As a comparative study, we evaluate the performance of GreenMap relative to two baseline methods: (1) the *no-approx* approach in which no approximation is performed on the processing tasks executed on the mobile device and (2) GraMap [5] in which the voxel grid and normal-based filters are performed on the server side at the cost of consuming more energy on the mobile device for data communication. The comparative study is performed in terms of the energy overhead on the mobile devices and the modeling accuracy of the generated indoor models.

B. Filtering Quality

In this section, we discuss the performance of our approximated filters for reducing the redundancy and irrelevant points by providing several examples.

1) DyPR: During our experiments, we found that DyPR outperforms the GraMap approach where it has 42.7% less processing time with 76.5% less data reported to the server. To achieve those results, DyPR exploits the sorting process performed by the *voxel grid filter* to have neighboring points next to each other and layer on layer bottom up. Nevertheless, this feature comes at the expense of some accuracy penalties where few holes on the wall segments are caused by false positives. To clarify this phenomenon, Figure 7 visualizes five subsegments of a single wall to demonstrate the effect of different filters. Through comparing Figures 7b and 7c which show two parts of a wall segment, we found that DyPR has a similar filtering behaviour as the non-approximated filter, except for the presence of a horizontal cut in the wall segment. These holes emerge due to the existence of furniture, e.g. tables and chairs, that increases the probability ω. Accordingly, the prediction algorithm erroneously classifies the wall points, located in the same height as the furniture's horizontal surface, as a non-wall points, i.e. false positives (cf. Figure 7d).

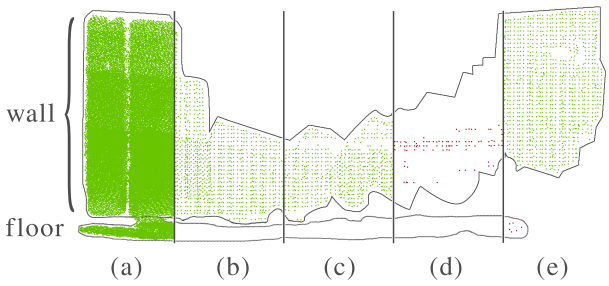
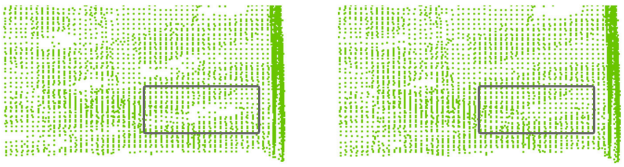


Fig. 7: Comparing the different filters, applied to different parts of a wall segment: (a) original wall and floor segment, (b) voxel grid and normals-based filters, (c) DyPR, (d) false positives in DyPR, and (e) SuFFUSION

2) SuFFUSION: In contrast to DyPR, SuFFUSION represents a deterministic approach that does not filter out points without examining them. Figure 7e shows that SuFFUSION has notably better results than DyPR with filtering behaviour comparable to the non-approximated filter. To examine the effectiveness of our adaptive threshold, Figure 8 compares the result of adopting SuFFUSION with and without the adaptive threshold. Obviously, Figure 8a comprises several holes which disappear once the adaptive threshold is applied while implementing the filter (cf. Figure 8b). Performing several experiments with different point clouds have proven

the robustness of SuFFUSION to false positives. Compared to GraMap, SuFFUSION achieves less processing time (at least by 60.3%) with reporting less data (at least by 79%). Furthermore, SuFFUSION outperforms the non-approximated filter where it achieves less processing time (at least by 48%) and less data is uploaded to the server (by at least 9%) without considerable quality losses.



(a) Without threshold tolerance (b) With threshold tolerance

Fig. 8: A wall segment filtered using SuFFUSION: (a) without and (b) with threshold tolerance highlighting missing points.

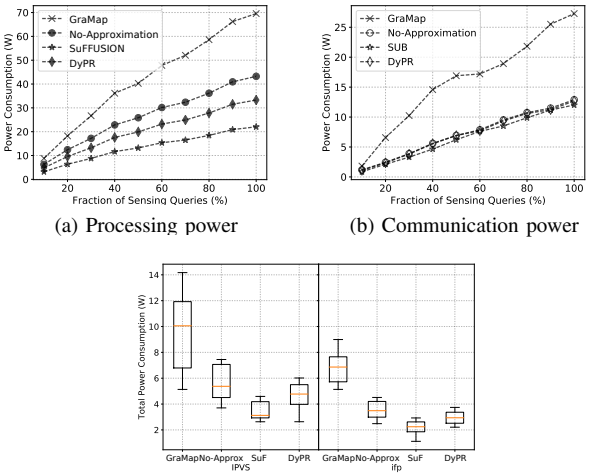
C. Power Consumption

In this section, we evaluate the energy consumption of data processing and communication on the mobile devices using our proposed GreenMap approach and the baseline methods. To measure the energy consumption on the mobile devices, we used the Android smart battery interface for energy consumption monitoring through logging the battery's current and voltage [18]. These values associated with their timestamp were recorded 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 consider only the energy overhead of running the algorithms. Each run of these measurements were repeated ten times and the resultant values are then averaged.

Figure 9a shows the cumulative power consumption consumed while processing the IPVS data set using the four compared approaches. The horizontal axis denotes the fraction of data processed on the mobile devices where 22 point clouds were sufficient to model the IPVS floor plan[1]. Obviously, the No-Approximation approach consumes less power than GraMap (at least by 37%). Our two approximation strategies DyPR and SuFFUSION reduce the power consumption relative to GraMap by 51% and 67.8%, respectively. Similarly, Figure 9b depicts the cumulative power consumption of communication. Again, GraMap consumes much more energy than our proposals where the No-Approximation, DyPR, and SuFFUSION consumed less energy by at least 55%, 58%, and 56%, respectively. We notice that our approximation strategies have similar communication power consumption thanks to their comparable filtering capabilities.

Finally, Figure 9c shows a comparison between the total power consumption of adopting the proposed approaches on several data bundles collected from both buildings. The behaviour of the four approaches is similar in both data sets where SuFFUSION has the lowest power consumption in both cases (on average 3.5 Watt per bundle for the IPVS data set and 2.2 watt per bundle for the ifp data set). From these results, we conclude that SuFFUSION and DyPR broadly improve the energy efficiency of the smart mobile devices while participating in data acquisition.

[1]Several areas, e.g. maintenance rooms were overlooked during data collection since they were not accessible



(a) Processing power (b) Communication power

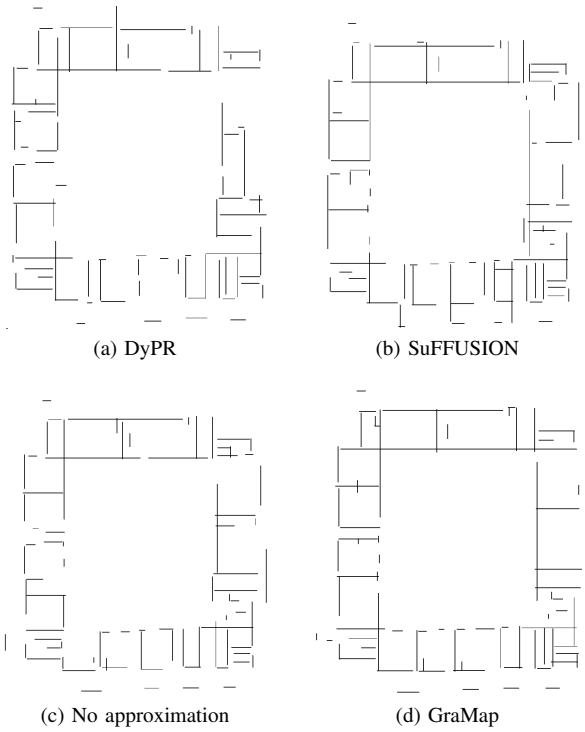
(c) Total power consumption in both data sets

Fig. 9: Comparison of the power consumption

D. Modeling Accuracy

In this section, we evaluate the accuracy of the generated floor plan after adopting the server-side processing. Our primary objective is to examine the impact of approximation on the modeling accuracy. To this end, we compare the initial and final indoor models obtained using different point clouds which have been processed using our two approximated filters and the baseline methods. We begin the comparative study with Figure 10, which depicts four initial indoor models of the IPVS floor plan generated using the two proposed methods and the two baseline methods. To generate these initial models, we divided the point cloud into a set of segments before projecting them onto the xy-plane (cf. Section V). As it can be seen in the figure, DyPR and SuFFUSION generate similar models compared to GraMap and the no-approximation approach.

To clearly quantify the difference between the compared approaches, we estimate the *room size error* as the Euclidean distance between the detected rooms size and the actual rooms size obtained from the ground truth. Figure 11 compares the error values of DyPR, SuFFUSION, and the GraMap approach while generating the indoor model of the IPVS and the ifp floor plans. For IPVS, we notice that SuFFUSION performs approximately similar to the GraMap approach where the average error of SuFFUSION is 0.51m compared to 0.46m for the GraMap approach. In contrast, DyPR has relatively higher modeling error than GraMap where its average error is 0.96m. Apparently, around 50% of the detected rooms in the three methods have an error value of less than 0.5m for IPVS and one meter for ifp. In both buildings, DyPR results in several outlier values above two meters, owing to the holes problem mentioned in Section VI-B.



(a) DyPR (b) SuFFUSION

(c) No approximation (d) GraMap

Fig. 10: Initial indoor models (no grammar)

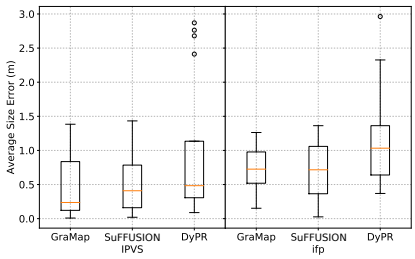
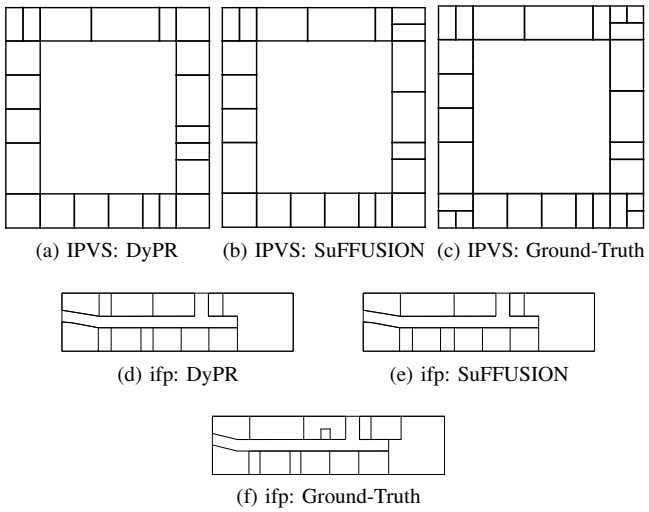


Fig. 11: Modeling accuracy of the proposed approaches

Apparently, the four initial models still suffer from inaccuracies and irregularities owing to the deficiencies of collecting point clouds from non-experts, i.e. quick recording with insufficient number of points and/or incorrect orientation of the mobile devices while recording. Therefore, the server utilizes the initial model as an input to a Hidden Markov model together with the formal grammar to generate a highly-accurate indoor model [5]. Figure 12 compares the generated final indoor models of the IPVS and the ifp floor plans with the ground truth. Thanks to the data extracted from the initial models and the formal grammar, SuFFUSION and DyPR are broadly similar to the ground truth except at the corner spaces where our current implementation of the grammar overlooks the generation of overlapping rooms, i.e. horizontal rooms intersect with vertical ones. For IPVS, SuFFUSION achieves a detection accuracy—defined as the ratio of the detected rooms relative to the actual number of rooms—of 90.5% while DyPR achieves an accuracy of 76.2%, excluding the rooms missing due to either the incomplete input data or the implementation of our formal grammar. Similarly, SuFFUSION achieves a detection accuracy of 91.3% relative to the ground truth while DyPR has a slightly smaller accuracy of 89.5% owing to the occurrence of several false positives and false negatives. From these results, we conclude that SuFFUSION and DyPR provide a similar accuracy as the GraMap approach even with approximate processing on the mobile devices. However, the results show that SuFFUSION outperforms DyPR owing to the holes problem and the probabilistic nature of DyPR.



(a) IPVS: DyPR (b) IPVS: SuFFUSION (c) IPVS: Ground-Truth

(d) ifp: DyPR (e) ifp: SuFFUSION

(f) ifp: Ground-Truth

Fig. 12: Final indoor models (with grammar)

VII. Related Work

In this section, we review the most salient related work from two relevant research disciplines—approximate computing and indoor mapping based on crowdsensing, with highlighting the novelty of our proposed solution.

A. Approximate Computing

In the past years, the potential of approximate computing has been explored in several works to reduce the operational costs via trading-off accuracy and other QoS metrics such as energy consumption and latency [8], [19], [20]. For instance, Li et al. [20] introduce MobiQoR, a system that jointly optimizes the offloading decisions and the selection of edge nodes' QoS level. To this end, the multi-objective optimization task is mapped onto a linear programming problem. Solving this problem determines the assignment of workload and the QoS level for each mobile edge node in terms of the energy consumption and the latency. Other approximation approaches focus on the computations performed on the mobile devices. These approaches typically analyze the online processing steps to identify the approximable parts. In this realm, Pandey et al. [19] propose MobiDic, a generic framework based on approximate computing for QoS-aware applications. To simultaneously shorten the execution time and reduce the energy consumption, Pandey et al. construct several workflows that dynamically utilize substitution and/or discarding as the approximation strategy. Alternatively, Dibak et al. [8] proposed a dedicated approximate computing framework for mobile simulations whose methods are specific to numeric simulations. Along a similar line, GreenMap introduce two specific

approximation strategies to improve the energy efficiency of mobile devices while crowdsensing 3D point clouds.

B. Indoor Mapping

Another line of related work is focusing on indoor models using crowdsensed data. In fact, mobile devices have been recently exploited to collect mapping data owing to being ubiquitous and equipped with relevant sensors such as accelerometers, magnetometers, and cameras [2]–[5]. In [2], we introduce MapGENIE, an architectural framework for automatically generating indoor maps supported by design-time knowledge, i.e. structural information. MapGENIE follows a two-stage approach. First, it adopts dead reckoning to derive an initial floor plan from a set of inaccurate motion traces. Afterwards, the inaccurate traces are refined using information about the building exterior encoded in a formal grammar. Similarly, Qiu and Mutka [3] present iFrame, an opportunistic approach that uses crowdsensed motion traces, Bluetooth fingerprints, and WiFi fingerprints to generate indoor maps. The core idea behind iFrame is to rectify the deviations of dead reckoning through curve fitting fusing. Alternatively, GraMap [5] exploits a new wave of smartphones which are capable of directly generating point clouds, such as Google Tango and Apple ARKit. In GreenMap, we further improve the energy efficiency of indoor modelling on mobile devices through exploiting the potential of approximate computing.

VIII. Conclusion & Future Work

In this paper, we introduced GreenMap, an energy-aware crowdsensing framework for collecting and processing point clouds to automatically generate indoor models. To this end, we proposed two approximation strategies, namely DyPR and SuFFUSION, to reduce the energy consumption while processing point clouds. The former strategy employs a probabilistic model to decide which points to process. Alternatively, SuFFUSION combines the operations of two filters so that heavy computations are entirely sidestepped. The results showed that DyPR and SuFFUSION reduce the energy overhead by 51% and 67.8% relative to GraMap, respectively. Considering the false positives problem of DyPR, we found that SuFFUSION is the most efficient approximation strategy thanks to its filtering behaviour. At the server side, we exploit formal grammars together with a HMM model to generate highly-accurate indoor models. Again, the results showed that SuFFUSION has a similar modeling accuracy as the baseline method. To sum up, the concepts used in SuFFUSION are highly promising to be transferred to other applications in order to optimize the online processing on mobile devices.

So far, we crowdsensed 3D point clouds to generate 2D floor plans. In future work, we plan to extend the SuFFUSION approach to be used for 3D models, especially for object recognition, e.g. furniture. In contrast to DyPR, SuFFUSION preserves all normals, and hence it does not limit the filter to 2D modeling. In this realm, SuFFUSION can be efficiently applied along with machine learning algorithms to extract useful information about the various 3D objects.

References

[1] F. Restuccia, N. Ghosh, S. Bhattacharjee, S. K. Das, and T. Melodia, "Quality of information in mobile crowdsensing: Survey and research challenges," *ACM Transactions on Sensor Networks (TOSN)*, vol. 13, no. 4, p. 34, 2017.

[2] D. Philipp, P. Baier, C. Dibak, F. Dürr, K. Rothermel, S. Becker, M. Peter, and D. Fritsch, "MapGENIE: Grammar-Enhanced Indoor Map Construction From Crowd-Sourced Data," in *Proceedings of the 2014 IEEE International Conference on Pervasive Computing and Communications*, ser. PerCom, 2014, pp. 139–147.

[3] C. Qiu and M. Mutka, "iFrame: Dynamic Indoor Map Construction through Automatic Mobile Sensing," in *2016 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, March 2016, pp. 1–9.

[4] R. Gao, M. Zhao, T. Ye, F. Ye, G. Luo, Y. Wang, K. Bian, T. Wang, and X. Li, "Multi-story indoor floor plan reconstruction via mobile crowdsensing," *IEEE Transactions on Mobile Computing*, vol. 15, no. 6, pp. 1427–1442, June 2016.

[5] M. Abdelaal, F. Dürr, K. Rothermel, S. Becker, and F. Fritsch, "Gramap: Qos-aware indoor mapping through crowd-sensing point clouds with grammar support," in *Proceedings of the 14th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Ser-vices*, ser. MOBIQUITOUS 2017. Melbourne, Australia, ACM, 2017.

[6] M. Abdelaal, D. Reichelt, F. Dürr, K. Rothermel, L. Runceanu, S. Becker, and D. Fritsch, "Comnsense: Grammar-driven crowd-sourcing of point clouds for automatic indoor mapping," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 1, p. 1, 2018.

[7] R. Schnabel and R. Klein, "Octree-based point-cloud compression," in *Proceedings of the Symposium on Point-Based Graphics*, M. Botsch and B. Chen, Eds. Eurographics, Jul. 2006.

[8] C. Dibak, A. Schmidt, F. Dürr, B. Haasdonk, and K. Rothermel, "Server-assisted interactive mobile simulations for pervasive applications," in *Proceedings of the 2017 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, March 2017, pp. 111–120.

[9] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. ACM, 2011, pp. 124–134.

[10] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys*, vol. 48, no. 4, pp. 62:1–62:33, Mar. 2016.

[11] I. Jolliffe, *Principal Component Analysis*. Wiley Online Library, 2002.

[12] ATAP, "Google project tango," 2016, accessed on June 2016. [Online]. Available: https://developers.google.com/project-tango/

[13] R. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *Proceedings of the 2011 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 1–4.

[14] W. R. Gilks, S. Richardson, and D. Spiegelhalter, *Markov chain Monte Carlo in practice*. CRC press, 1995.

[15] A. Nguyen and B. Le, "3d point cloud segmentation: A survey," in *Proc. of the Conf.on Robotics, Automation and Mechatronics (RAM)*. IEEE, 2013, pp. 225–230.

[16] S. Becker, M. Peter, D. Fritsch, D. Philipp, P. Baier, and C. Dibak, "Combined grammar for the modeling of building interiors," *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. II-4/W1, pp. 1–6, 2013.

[17] G. T. Developers, "Integrating motion tracking with area learning," 2016, accessed on May 2016. [Online]. Available: https://developers.google.com/tango/overview/area-learning

[18] Q. Nguyen, J. Blobel, and F. Dressler, "Energy consumption measurements as a basis for computational offloading for android smartphones," in *2016 IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, Aug 2016, pp. 24–31.

[19] P. Pandey and D. Pompili, "Mobidic: Exploiting the untapped potential of mobile distributed computing via approximation," in *Proceedings of the 2016 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, March 2016, pp. 1–9.

[20] Y. Li, Y. Chen, T. Lan, and G. Venkataramani, "Mobiqor: Pushing the envelope of mobile edge computing via quality-of-result optimization," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 1261–1270.