Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit

# Design and Implementation of an Indoor Modeling Method through Crowdsensing

Daniel Reichelt

| | |
|---|---|
| **Course of Study:** | Softwaretechnik |
| **Examiner:** | Prof. Dr. Kurt Rothermel |
| **Supervisor:** | Dr. Mohamed Abdelaal |
| **Commenced:** | 2016-10-01 |
| **Completed:** | 2017-03-31 |
| **CR-Classification:** | C.2.4, I.4 |

# Abstract

While automatic modeling and mapping of outdoor environments is well-established, the indoor equivalent of automated generation of building floor plans poses a challenge. In fact, outdoor localization is commonly available and inexpensive through the existing satellite positioning systems, such as GPS and Galileo. However, these technologies are not applicable in indoor environments, since a direct line of sight to the satellites, orbiting the globes, is required. As a substitution, the technical literature comprises several proposals for the development of simultaneous indoor localization and mapping (SLAM). In these approaches, the authors mostly exploit indoor resources such as the WiFi access points and the mobile smart devices carried by individuals in the indoor environment. Collecting data from several mobile devices is referred to as *crowdsensing*.

To enable the generation of two-dimensional (2D) as well as three-dimensional (3D) maps, we propose crowdsensing of point clouds, which are 3D data structures of points in space. For localization, we integrate two features of a recently developed mobile device, called *Project Tango*. Specifically, the Tango platform provides two main technologies for reliable localization, namely *motion tracking* and *area learning*. Moreover, Tango-powered devices provide us with the ability to collect point clouds though a third technology, called *depth perception*. In the past few years, spatial data obtained from range imaging was used to generate indoor maps. Nevertheless, range images are expensive and not always available. The required equipment, e.g. laser range scanners, are both expensive in procurement and require trained personnel for proper setup and operation.

In this thesis, we aim for obtaining spatial point clouds via crowdsensing. The main idea is to use sensor data which can be scanned by volunteering individuals using easy to handle mobile devices. Specifically, we depend on depth perception capabilities as provided by Google Tango-powered tablet computers. A crowdsensing infrastructure assigns scanning tasks to individuals carrying a Tango device. Execution of such a task consists of taking scans of e.g. offices in a public building. The scanning results contain both spatial information about the room layout and its position. Energy consumption on the mobile device is reduced by applying Octree compression to the scanned point clouds, which results in a significant reduction of the amount of data, which has to be transferred to a back-end server.

Afterwards, the back-end is responsible for assembling the received scans and the extraction of an indoors model. The modeling process – developed in this thesis – comprises two-phases. First, we extract a basic model from the obtained point clouds, which may contain outliers, inaccuracies and gaps. In the second phase, we refine the

model by exploiting formal grammars. It is worth to mention here that we are the first to exploit formal grammars as a model fitting tool. We feed the information obtained in the first phase to an indoors grammar, which has been developed in the ComNSense project, University of Stuttgart. The resultant model both contains much less deviations from the ground truth and provides improved robustness against aberrations with respect to localization during the scanning process. Thus, instead of scanning multiple point clouds per room, we need only one scan to be able to construct an indoor map. During evaluation of this process, using scans of offices of our department, we were able to reproduce a model which is very close to the ground truth.

## Acknowledgments

I would like to thank Prof. Dr. Kurt Rothermel for his support in my search for a thesis subject in his department, and giving me the opportunity to work on this topic.

A very special gratitude goes to my supervisor, Dr. Mohamed Abdelaal. Although he keeps saying, he did only his job, I would like to say thank you for many hours of interesting discussions, our sharing of ideas, his truly always open ears for any problem, and his constant optimism and moral support.

Also a special mention to my dear friend, Julian Trischler, for your input.

And finally, but by no means least, to my family, for keeping my back free during this – not always stress-free – time.

Thank you!

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

**ARM** Microprocessor Architecture, also Advanced RISC Machines Ltd.. 56

**CPU** Central Processing Unit. 56

**CSV** Comma-Separated Values. 65

**CUDA** Compute Unified Device Architecture. 77

**EV** Eigenvector. 44

**GCC** GNU C Compiler. 57

**GPU** Graphics Processing Unit. 77

**NDK** Native Development Kit. 56, 57

**OBB** oriented bounding box. 44

**OOP** Object-oriented Programming. 40

**PCA** Principal Component Analysis. 44

**PCL** Point Cloud Library. 41, 42, 55, 63, 64

**QoS** Quality of Service. 19

# 1 Introduction

## 1.1 Motivation

A large variety of systems and applications are offering localization and navigation services for outdoor environment. However, people typically spend most of their time in indoor environments. Hence, it appears the need for updated indoor maps especially for public spaces. The majority of current research endeavours focus on the indoor positioning problem [LDBL07; LLY+15; Mau12]. Obtaining spatial information in indoor environments is a challenge. Inertial sensors, accelerometers and gyroscopes, which are available on most smartphones, are used for positioning. The barometers built into smartphones are used to estimate the vertical position or to recognize vertical movements. The magnetometers built into smartphones are used for attitude estimation as part of a positioning system. Indoor positioning using WiFi signals is heavily investigated. The common approach to position estimation in these works is based on the received signal strength (RSS) and the fingerprinting method.

On the one hand, indoor positioning is currently affordable using recent commercial technologies. The indoor equivalent floor plans are currently very limited, affecting the ubiquity and spread of indoor location-based applications. On the other hand, there still is a gap in developing reliable indoor maps. Recently, Google Maps started to provide detailed floor plans for a few malls and airports in the U.S. and Japan. Nevertheless, all these systems depend on manually building the floor plan. Manual addition/editing of all buildings' floor plans around the world requires an enormous cost and effort which may be unaffordable. In addition, keeping these floor plans up to date is another challenge.

Considering that normally the indoor mapping is an expensive procedure in terms of needed instruments and specialized personnel, alternative ways of automatic indoor mapping are recently a subject of research. An innovative solution for this issue is to exploit the data that is already available, coming from mobile devices [BPDR14]. Despite having become an everyday life commodity, they are a meaningful source of information supported by powerful sensors and increasing computational power. In this way, this thesis has the purpose of using the mobile data coming from the crowd, for automatically mapping indoor environments.

In this thesis, we target developing an efficient and automatic floor plan construction system. Such a system leverages the ubiquity of mobile smart phones to infer information about the building floor plan. As aforementioned, today's smart devices have an array of sensors, e.g. inertial sensors (accelerometers, compasses, and gyroscopes), that can be used to construct traces of movement in a transparent manner to the users. Using this crowdsensing approach, we can provide the general layout of a building, identify rooms, corridor locations and shapes, along with identifying other points of interest, such as elevators, stairs, and escalators.

Several research articles have been devoted to deriving indoor models. For instance, Damien et al. [PBD+14] propose an automatic method for deriving the indoor model via collecting odometry traces. They later enhanced the obtained model via considering a formal interiors grammar. Chen et al. [CLRQ15] introduce a new automatic method, referred to as CrowdMap. CrowdMap jointly leverages crowdsensed sensory and video data to track user movements. Afterward, it uses the inferred user motion traces and context of the image to produce an accurate floor plan. Alzantot et al. [AY12] propose an interior model generation, called CrowdInside, which leverages the ubiquity of smart phones to infer information about the building floor plan along with other semantic information. They collect measurements from sensors including: accelerometers, magnetometers, gyroscopes, and the received WiFi signal strength values from available access points. Similarly, Qiu et al [QM16] introduce an indoor map construction method, referred to as iFrame. In particular, iFrame combines dead reckoning and RSSI detection techniques to judge whether the subareas in an indoor environment are empty. However, all these approaches utilize traditional observations as the source of semantic information. These observations are useful only if a 2D indoor model is required. This means that 2.5D and 3D models are not feasible with these observations.

To tackle this problem, several research articles proposed point clouds to be the source of geometrical information. For instance, Budroni et al. [BB10] introduce a fully automated 3D modeling method of indoor environments from point clouds. The point clouds were acquired using a laser scanner with several scans. Afterward, the collected point cloud data is processed in order to segment planar structures, which have a noticeable architectural meaning (floor, ceiling and walls) in the interior. Referring to crowdsensing, the utilization of laser scanners hinders collecting data in a distributed fashion. Additionally, these laser scanners are relatively expensive.

As a solution to these challenges, we select to exploit a recently-developed technology by Google Inc., referred to as *Project Tango* [16a]. Tango capabilities give mobile devices a human-scale understanding of space and motion. To this end, Tango-powered devices embed three novel features, including *depth perception, motion tracking*, and *area learning*. Motion tracking means that a Tango-powered device can track its own movement and orientation through 3D space. This means that Tango-powered devices are capable of detecting various movements such as forward, backward, up, down; or being tilted in any direction. Depth perception enables mobile devices to understand the distance to objects in the real world. Finally, area learning gives the mobile device the ability to see and remember the key visual features of a physical space – the edges, corners, other unique features – so it can recognize that area again later.

In this thesis, we exploit an indoor mapping model via crowdsensing point clouds. Tango-powered devices are used to collect the point clouds. We make use of the three Tango features to improve the modeling accuracy. For instance, we integrate area learning with motion tracking to correct the continuous and growing drifts. Additionally, we integrate area learning with depth perception to unify the reference to all collected point clouds. After data acquisition, we apply several filtering stages in order to keep only the segments, representing walls.

As a contribution of this thesis, we apply formal interiors grammars to enhance the modeling accuracy and to reduce the overall energy consumption. In general, grammars are a powerful tool for precisely describing and analyzing objects such as languages, behaviours, buildings, etc. In the project ComNSense [13], we developed a formal grammar to describe public buildings. Specifically, the developed grammar comprises rules for describing different room types and probabilities of transition between these rooms. We employ the interiors grammar to fit the model obtained via processing the point cloud data. Moreover, we examine the ability of the grammar to reduce the number of scans required to model the building interiors. Below, we give an general overview of crowdsensing, before we express the thesis' contributions in more detail.

## 1.2 Crowdsensing Overview

Literally, crowdsensing is a new technology to build low-cost wireless networked sensors. The core idea behind crowdsensing systems is to exploit the powerful capabilities and features of modern mobile devices to collect information about their environment. In particular, smart mobile devices are currently indispensable from our daily routine. These mobile devices exist everywhere and cover a large geographical area where they can inexpensively report the sensed data by means of the existing wireless network infrastructure such as WiFi and 3G networks. Originally, Abdelzaher et al. [AAB+07] first proposed the concept of crowdsensing systems. Since this date, several research work was devoted to developing new crowdsensing applications and to optimizing the associated Quality of Service (QoS) metrics. Nowadays, most mobile devices are equipped with a large variety of sensors such as accelerometers, proximity sensors, microphones, magnetic field sensors, GPS senors, and cameras.

Furthermore, some vendors equip special sensors. For instance, a new model of Apple's iPhone comes with smart sensors such as barometers, gyroscopes, BLE modules, etc. Similarly, wearable devices, such as smart watches and smart glasses can be easily integrated into the crowdsensing systems. Due to their powerful capabilities, the crowdsensing systems can be applied in different areas, like environmental monitoring, advertising, indoor mapping applications, and traffic monitoring. Crowdsensing opens the door for developing new powerful and smart applications and services for both big and small companies. Crowdsensing enables an application to make much more informed decisions and interact with the users based on data that is gathered in real-time.

Crowdsensing systems can principally be categorized according to the human involvement into *opportunistic sensing* and *participatory sensing* [Bai15]. In the former class, the back-end servers passively collect sensor data without assigning data collection tasks to the mobile devices owners. Thus, gathering sensor data is a background operation, such as position fixing and detection of orientation. The second class of crowdsensing applications is a participatory sensing approach; sensor data is typically collected through interacting with the mobile device users. A crowdsensing system asks the user by sending a request to a running application on the mobile device that can process the request and respond to it. In this thesis, we leverage the participatory sensing mechanism to collect the required point clouds.

A typical crowdsensing system comprises a set of participating mobile devices, servers, communication networks, and client applications, as depicted in Figure 1.1. A crowdsensing server is responsible for serving all queries from different applications in the system. Moreover, it manages the sensing query, controls the sensing processes and coordinates the communication between the served clients and the participating mobile devices. Specifically, the crowdsensing server has a *query interface* which makes the communication between a client and a server easier and more flexible. The query interface is used by a client to send queries and to receive the required data. In addition, a *basic sensing system* is included to manage the received queries. It decodes the received

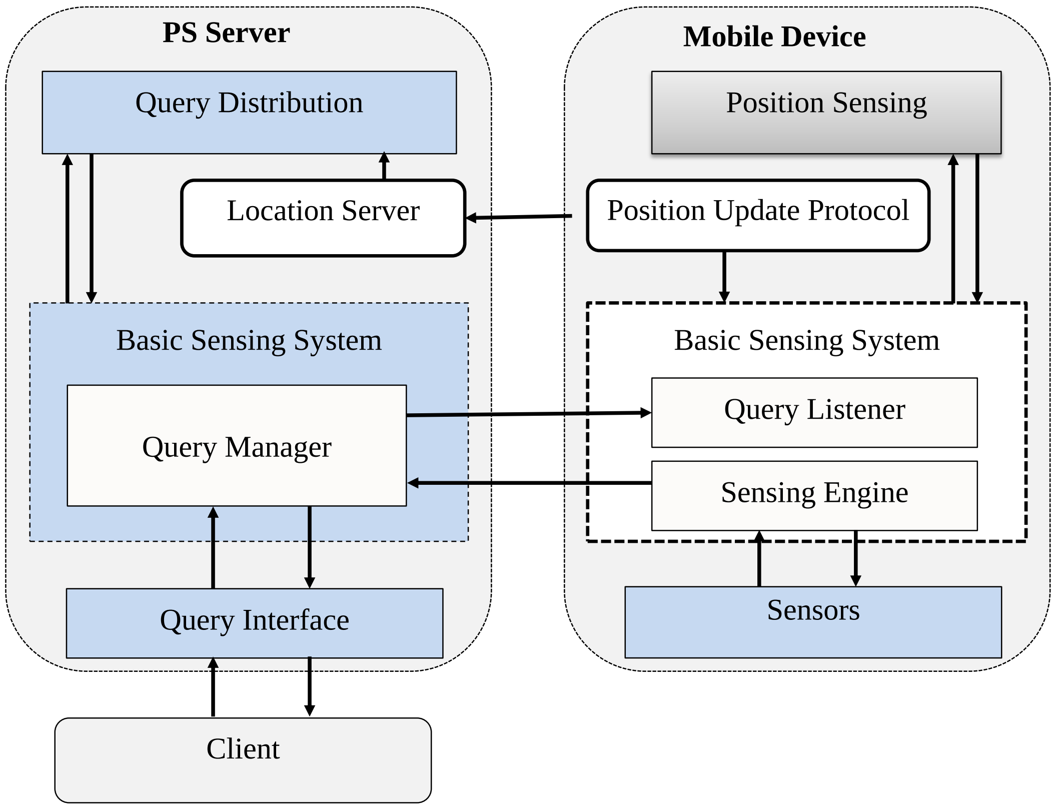


**Figure 1.1:** General architecture of crowdsensing system [Bai15]

queries and then forwards it to the set of participating mobile devices. To this end, a *position update protocol* keeps track of the mobile devices in the sensing area.

Similarly, the participating mobile devices run a basic sensing system which is responsible for managing the received sensing queries. Specifically, it comprises two sub-units, *query listener* and *sensing engine*. The query listener waits for any query and forwards it to the sensing engine. Afterward, the sensing engine manages the sensing process by controlling the sensors. For example, activating or deactivating a sensor to scan an area to collect motion traces and point clouds. The sensing engine answers the query by sending the requested sensor data back to the server. In addition, an update position protocol frequently senses the position of the mobile device and sends an update to the corresponding server.

There are several QoS requirements for the crowdsensing systems, including privacy, coverage, scalability. For instance, the crowdsensing system should guarantee that the whole geographic monitoring area is covered by leveraging the availability of the mobile devices. Moreover, it should guarantee the quality of the gathered data, i.e. prevent undesired impacts of unreliable data sources. Furthermore, it should motivate and attract the users so that they accept to participate in the data gathering process. However, energy consumption is considered as the most important criterion while designing a crowdsensing system. The reason revolves around the users' acceptance to participate in the crowdsensing system. If the additional tasks – due to participating in the mobile crowdsensing system – consumes an excessive amount of energy, then users typically lose their motivation to share their resources for collecting data. To ensure a user's acceptance, the crowdsensing systems have to be highly energy efficient so that the energy resource on the participant's mobile device is preserved. Another important QoS metric is the user privacy which has to be considered while designing the system.

## 1.3 Indoor Mapping

The availability of maps is highly important while dealing with location-based services. Literally, a map is typically defined as a model that describes threefold features, namely *topology*, *geometry* and *semantics* [Ind16]. Topology defines which areas are connected and which areas are not. Geometry defines objects which are represented by means of coordinates in a reference system. The simplest spatial object is a single point expressed by its x, y and z coordinates, e.g. two endpoints model a straight line. Similarly, an ordered list of points models a polyline. A polygon is defined when the start node and end node of a polyline match. Finally, semantics indicates the way that the space can be used (e.g. stairs, elevator, etc.) as well as unique identifiers of the place, e.g. the received signal strength in a room from multiple access points.

Recently, OpenStreetMap is released as a crowdsensing project with the goal of creating a free map of the world [Ope16]. In this project, volunteers record motion traces via their smartphone's built-in GPS sensor. The resulting traces are combined to remove measurement errors and to determine the type of path traversed by the users. Although these approaches reliably work for outsdoor settings, they are not suitable for indoor environments. This limitation emerges due to the lack of absolute positioning systems, such as GPS or Galileo. Therefore, indoor models were created in the past manually through converting building blueprints or by tediously mapping buildings using 3D laser scanners. However, such manual generation of indoor maps typically consumes too much time and effort.

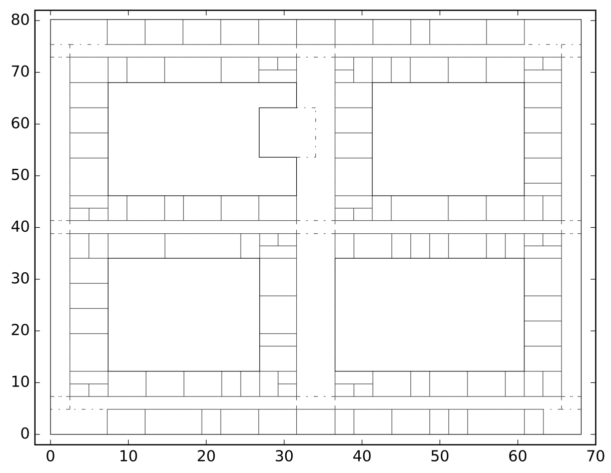


**Figure 1.2:** Ground truth of the floor layout

Since recently, several efforts are being made for automatically deriving indoor maps. Crowdsensing is exploited in the realm to mitigate the overhead of collecting data. Figure 1.2 depicts the ground truth layout of the second floor of the Computer Science Department, University of Stuttgart, Stuttgart. We seek – through this thesis – to automatically generate a map close to the one shown in Figure 1.2. As depicted in the figure, the floor layout is composed of exactly four quadrants. Since these quadrant are similar, we seek to collect data from the quadrant which is easily accessible by our volunteers. To this end, the bottom left quadrant, which comprises the employee's offices of our department was selected as target area for sampling point clouds. For evaluating the accuracy of our model, we determine the modeling error relative to the shown map, representing the ground truth.

## 1.4 Contributions

The main goal of this thesis is to design and implement an interior model derivation method. Crowdsensing exploits mobile devices existent in an indoor environment to collect a set of point clouds. In this thesis, we rely on the powerful features provided by the Google Tango tablet to generate a 2.5D interior model. For instance, Tango tracks the motion of particular device, and creates a 3D representation of the environment around it, also known as point clouds. It also includes development APIs to provide alignment, position or location, and depth data to regular Android apps. We collect a set of point clouds from different mobile devices to cover the entire building. Then, we follow a multistage filtration approach to keep only the walls.

Subsequently, we enhance the resultant model via adopting formal grammars. In the project ComNSense, we developed a formal grammar which encodes structural information. In fact, the 3D model – resultant after filtration steps – is not complete and the segments are usually not connected. To improve the modeling accuracy, we extract the semantic information from the point cloud-based model such as room width and the ceiling height. Subsequently, we feed this information to the grammar to generate a clean version of the model in which all lines are connected. In more detail, the thesis contributions are summarized as follows.

- Collecting point clouds – from the participants mobile devices in the crowdsensing system – is achieved via integrating depth perception and motion tracking with area learning.
- Development of a novel processing pipeline which comprises multi-stage point cloud filtering, including normals estimation and segmentation by region growing.
- Enhancing the modeling quality and energy efficiency is performed through incorporating the formal grammars.
- A comparative study between the proposed grammar-based method and the grammar-free model generation method is carried out.
- The performance of the proposed method is evaluated in terms of the energy overhead on the participating mobile devices (i.e. number of point clouds sampled per each device), the computational overhead of the method, and the model accuracy.

## 1.5 Document Structure

This thesis is structured as follows:

**Chapter 2 – System Overview:** This chapter describes the system model this work is based on and provides an overview of related work in the area of crowd sensing and indoor mapping.

**Chapter 3 – Basic Interiors Model:** In this chapter, we introduce a basic process to obtain a floor plan from crowd-sensed point clouds. Each step of the process is described along with a short introduction into the underlying concepts and algorithms.

**Chapter 4 – Grammar-Enhanced Mapping:** Introduction of interiors grammars and their combination with the basic model into an enhanced, grammar-based indoor mapping process is the main subject of this chapter.

**Chapter 5 – Implementation:** In this part, we introduce the software developed in the course of this work and other tools, which are used to drive the mapping process.

**Chapter 6 – Evaluation:** In a real-world application of both processes (basic and enhanced), we evaluate their performance in terms of effectiveness and accuracy.

**Chapter 7 – Conclusion and Future Work:** Finally, we briefly summarize this work and elaborate on some of the processes' current limitations and potential research topics to alleviate these shortcomings.

# 2 System Overview

In this chapter, we explain the system model before we formalize the attacked problem of reducing the energy overhead on the mobile devices and improving the modeling accuracy. Subsequently, we review examples of recent work in the realm of indoor mapping.

## 2.1 System Model

In this section, we provide an overview of the system architecture. Moreover, we describe our assumptions which are adopted throughout the entire thesis. As can be seen in Figure 2.1, the system model comprises three main components. First, a set of Tango-powered devices, which are carried by individuals, exist in the indoor environment, referred to as the *sensing area*. Second, a crowdsensing server collects the sensed data and coordinates the communication with the mobile devices. The servers usually send sensing queries to a set of users located in the sensing area. Finally, a mobile communication network allows the mobile devices to communicate with the crowdsensing system. Below, we discuss these components in more detail.

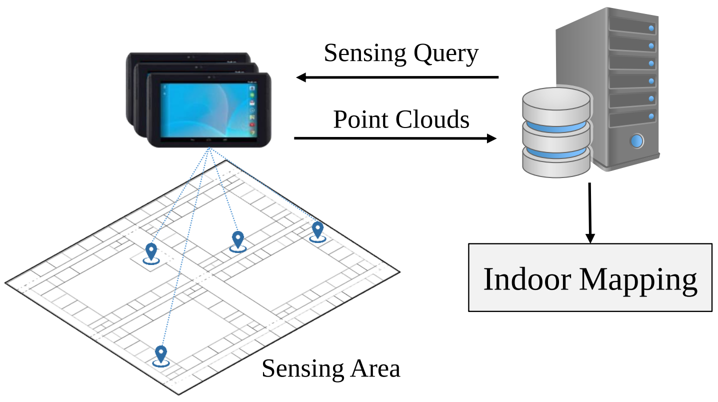


**Figure 2.1:** System architecture

### 2.1.1 Tango-Powered Mobile Devices

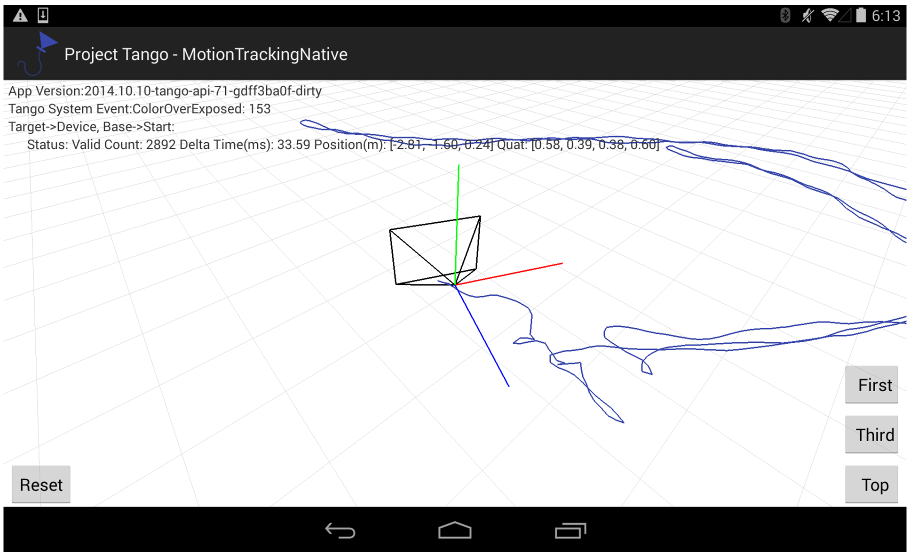
Project Tango is a device developed by Google's Advanced Technology and Projects (ATAP) group with the intention of adding human-like sensors to technology [16a]. Tango technology involves several new features, including motion tracking and depth perception sensors that allow the device to locate itself in space and compute its distance from objects that are in the range of sight of these sensors. Moreover, Tango enables area learning which represents a memory of the places which have been visited. The device offers a four megapixel RGB-IR rear-facing camera along with a 170 degrees fish-eye motion tracking camera that updates at 30Hz. The infra-red projector is also positioned in the rear of the device and has a refresh rate of 3Hz. The accelerometer, gyroscope and compass offer six degree-of-freedom positional data.

Unlike Kinect, which includes similar sensors [Mic17], Project Tango, just like other Android devices, offers on-board processing and storage. It has a 2.3 GHz quad-core NVIDIA Tegra K1 CPU and also offers 4 GB of RAM and 128 GB internal storage, which is expandable via microSD. Project Tango offers three Application Programming Interfaces (APIs), including: Java API, C API and the Unity API. We intend to exploit the three main features of Tango while deriving the floor plan. In the sequel, we describe each of the aforementioned features.

#### 2.1.1.1 Motion Tracking

Motion tracking is a new feature of Tango-powered devices which is based on the visual and the inertial odometry. This means that computer vision techniques are combined with measurements from the inertial sensors to achieve mapping of the environment. Moreover, it achieves positioning of the device within an indoor environment relative to the objects in the scene and tracks the device in the given space [16d]. Specifically, computer vision techniques involve feature matching – a technique used to match features (objects or part of them) between frames – and feature tracking, a technique which is used to track the matched features in order to determine the position of the camera/device relative to those features. Figure 2.2 depicts an example of tracking a user's movement using the motion tracking feature of Tango. The blue curve represents the path traversed by the user during this experiment.

Similarly, after the position of the device is established, the speed with which it is moving can be determined based on the change of position between frames. In particular, Google provides Tango APIs to collect the pose of the device, i.e. position and orientation, in



```
App Version:2014.10.10-tango-api-71-gdff3ba0f-dirty
Tango System Event:ColorOverExposed: 153
Target->Device, Base->Start:
    Status: Valid Count: 2892 Delta Time(ms): 33.59 Position(m): [-2.81, -1.60, 0.24] Quat: [0.58, 0.39, 0.38, 0.60]
```

**Figure 2.2:** An example of motion tracking

six degrees of freedom (6DOF) from the inertial sensors: accelerometer, gyroscope and compass. The returned data is a 3D vector in meters for translation and a quaternion for rotation. Both sources need to be fused in order to achieve tracking, but Google does not supply information on how they achieve data fusion for tracking purposes. In fact, motion tracking has some limitations, including:

- Drifts: after walking long distances for a period of time error is accumulated and the perceived position from the device will have drifted from the actual position the user is at.
- Motion tracking does not understand the space around it. Every time a new tracking session starts, it will be based on the latest starting position, not remembering anything from previous tracking sessions.

Both of these limitations can be sidestepped via integrating motion tracking with area learning. Below, we explain the area learning feature of Tango-powered devices.

#### 2.1.1.2 Area Learning

Area Learning is a process that enables a Tango-powered device to “learn” the area around it while keeping track of the user’s position in the given scene. To this end, Tango recognizes the space – in which the device is in – through matching what the device currently sees with what it has seen before. Specifically, area learning easily recognizes spaces that are visually distinctive, i.e. with corners and edges – in contrast to empty rooms with white walls. It also depends on the conditions under which the environment is being registered, i.e. day/night and furniture changes. Walking around the area, with area learning mode activated, enables the device to create an improved model of the area and thus correct drift. Area descriptions are saved in compressed form in *Area Description Files* (ADF) and can be reloaded whenever area learning is to be continued.

Figure 2.3 depicts a comparison of several motion traces collected while traversing the corridor of the computer science department, University building in Stuttgart. The red solid lines denote the user trajectories when only motion tracking was enabled. As can be seen, the traces are completely out of the correct path due to accumulated drifts. The black dotted lines represents walking in the same path but this time with area learning mode enabled and having loaded a previously stored ADF. Obviously, area learning significantly improves the accuracy of the collected traces. Drifts are corrected via remembering the visual features of the area it has visited, and uses them to correct errors in its understanding of its position, orientation, and movement. When a Tango device recognizes an area it knows, i.e. it has seen earlier in previous session, it realizes that the device has traveled in a loop and adjusts its path to be more consistent with its previous observations.

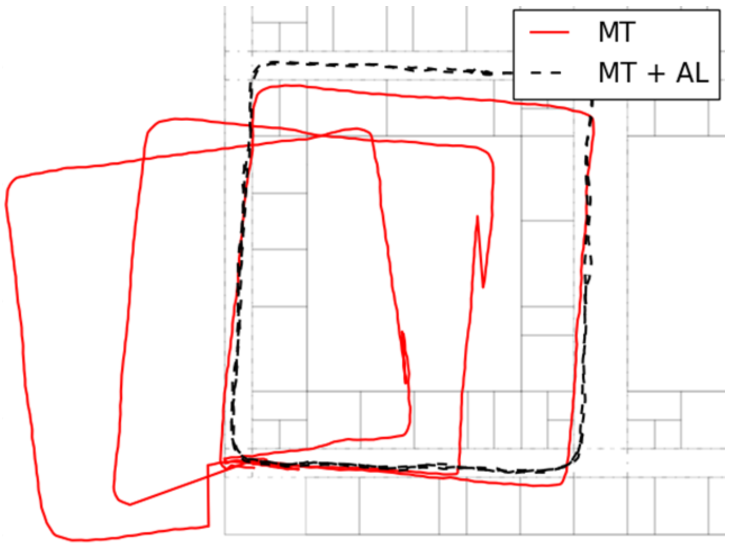


**Figure 2.3:** Impact of integrating motion tracking (MT) and area learning (AL)

According to Tango developers [16b], environmental changes which may occur in the indoor areas can degrade the performance of area learning. We observed this limitation during data collection where the ADF file was recorded during the day. Afterward, all collected point clouds and motion traces during the day were accurately positioned. On the other hand, the data collected at night using the same ADF file suffers from misplacement and drifts. To tackle such a problem, we advocate recording multiple ADFs for a single physical location under different conditions. This gives the participating mobile devices the option to select a file that most closely matches their current conditions. In the same regard, multiple sessions can be appended to the same ADF to capture visual descriptions of the environment from every position and angle and under variations of lighting or environmental change.

#### 2.1.1.3 Depth Perception

Depth perception enables Tango devices to estimate their distance from objects in the scene. Project Tango allows three approaches of achieving depth perception, namely *structured light*, *time-of-flight* and *stereo*. The first two approaches exploit the infra-red sensor to estimate the depth. In the time-of-flight approach, the distance from the objects will be estimated from the time it takes the infra-red waves to be reflected back and received at the infra-red sensor. Whereas, the third approach calculates the distance based on the pictures taken from two cameras, i.e. similar to the human eyes.

The depth APIs – provided by Project Tango's developers – return point clouds from the depth sensor in the form of $xyz$ coordinates as float values given in meters. There are two different coordinate systems, referred to by the developer documentation as "Right Hand Local Level" and "Right Hand Android" (cf. [16c]).

The Java API uses the "Right Hand Android" variant to describe coordinates and vectors for localization and orientation. However the final point clouds which are saved to files for further processing employ the "Right Hand Local Level" variant. We consider this an implementation detail of the scanning app and will use the "Right Hand Local Level" variant throughout this thesis, since all calculations for obtaining an interior model are based on this one.

The "Right Hand Local Level" coordinate system is defined as follows: If the device is in landscape orientation with the screen facing the user, then $+y$ points in the direction of the camera's optical axis and is perpendicular to the plane of the camera, $+z$ points toward the top of the screen and $+x$ points toward the user's right.

Specifically, the API returns coordinates of objects in space relative to the device, i.e. not direct distance measurements. The distance can be easily estimated as the Euclidean distance of two points in the 3D space. One limitation of the depth perception is that the surrounding sources of light and heat, which are infra-red sources, interfere with the depth sensor. Also, the depth sensors fail to detect transparent or reflecting surfaces since the waves travel through them or are diverted such that their reflection cannot be received by the sensor.

Figure 2.4 shows an example of a point cloud, collected using our Tango device. In this scan, we sweep the device to catch the distance between the device and the walls as well as the furniture. This figures shows the data which we can get from Tango devices, if it participates in a crowdsensing-based indoor mapping application. In this thesis, we collect similar point clouds and then apply several processing steps to identify interesting points only, e.g. walls.

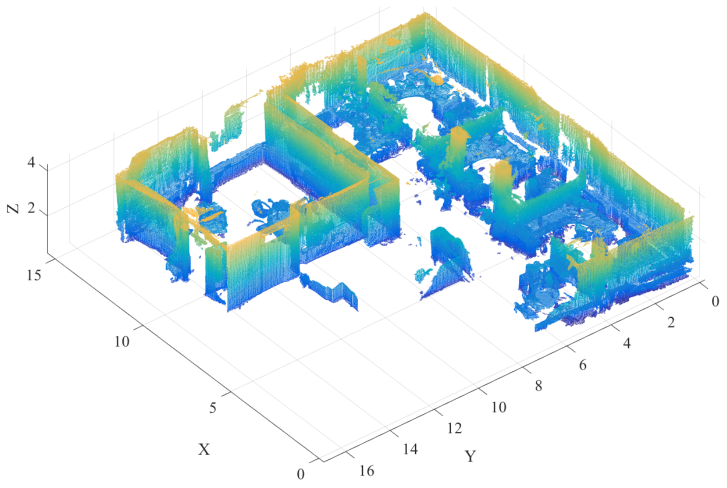


**Figure 2.4:** An example of point cloud collected using a Tango tablet

### 2.1.2 Crowdsensing Servers

The back-end servers represents an intermediate stage between the application layer and the data collection layer. They send sensing queries to the participating mobile devices. In this regard, several efforts have been exerted to identify only a subset of participating devices for the sake of improving the energy consumption. For instance, Baier et al. [BDR13] introduce a data quality model which relies on estimating the data completeness at the application layer. Based on the estimated data quality, the server selects a subset of the available mobile devices to which sensing queries are transmitted. In general, there may exist several crowdsensing servers, serving the same application, and being connected via a wired network. Each crowdsensing server is responsible for the mobile devices in a certain geographical area, which is referred to as the server's sensing area, as shown in Figure 2.1. For simplicity, and taking into account working in indoor environments, we consider – in this thesis – the case at which only one crowdsensing server exists. Accordingly, all participating mobile devices move within the sensing area of this server. Moreover, we assume that the sensing area is entirely covered by one server, i.e sensing queries are inside this area. Below, we describe sorts of communication between the back-end servers and the corresponding mobile devices.

### 2.1.3 Communication Facilities

Several wireless communication technologies exist for transferring the sensing queries and the sensed data between the crowdsensing servers and the participating mobile devices. For instance, this wireless communication can be done using either WiFi networks, or via cellular mobile networks such as UMTS or LTE. However, we assume that most modern buildings are completely covered by WiFi networks. Therefore, we only consider WiFi communication to convey data and queries between the crowdsensing components. Another reason for our selection emerges from the need to save as much energy as possible. Figure 2.5 demonstrates a comparison between WiFi and 3G in terms of the consumed energy when uploading three datasets. The datasets, referred to as *single*, *double* and *triple*, represent point clouds which result from scanning three different-sized rooms, ranging from small room, medium-sized room to large room. As can be seen in the figure, uploading the datasets using WiFi consumes much less energy compared to 3G. Therefore, we rely on WiFi for uploading the acquired point clouds to the crowdsensing server.

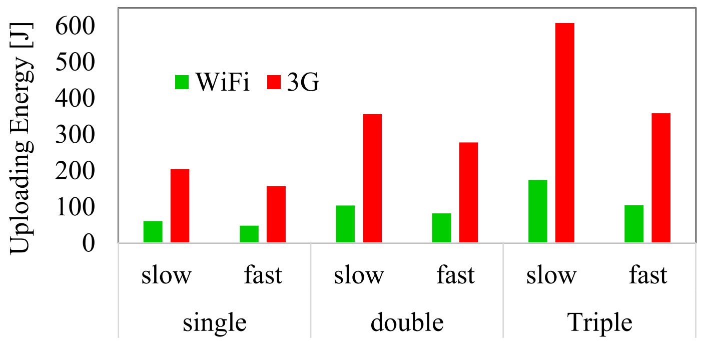


**Figure 2.5:** Uploading energy consumption in case of WiFi and 3G

## 2.2 Problem Statement

The problem with indoor mapping using crowdsensed point clouds is that point clouds are bulky enough to drain the mobile devices' batteries once they are processed or uploaded. Even with a WiFi connection between the server and the participating mobile devices, data transmission consumes an excessive amount of energy. In order to tackle this problem, we select to adopt a lightweight compression method which can reduce the uploading energy overhead via reducing the number of transmitted bytes. For this sake, we employ the *Octree compression* provided by the PCL library [RC11]. Another concern emerges due to compressing the point clouds prior to transmission where this compression may have negative impact on the modeling accuracy. In other words, compression may lead to miss important details of the point clouds. Therefore, we formalize the research problem as an *integer linear programming* problem to improve both of the energy consumption and the modeling accuracy, as denoted by Equation 2.1.

$$\begin{aligned} &\text{minimize} && \sum_{i=1}^{N} E_i(task), \quad task \in \{proc,\ upload\} \\ &\text{subject to} && \Omega(\mathcal{P}) = \Omega(\mathcal{M}_{\mathcal{P}}), \quad |\mathcal{M}_{\mathcal{P}}| << |\mathcal{P}|. \end{aligned} \tag{2.1}$$

Equation 2.1 defines the main objective as minimizing the overall energy consumption $E$ for the required tasks, including processing $proc$ and data uploading $upload$. This objective should be satisfied given that modeling accuracy does not degrade, i.e. modeling accuracy using the raw point cloud $\Omega(\mathcal{P})$ should equal that if only we use the compressed version $\Omega(\mathcal{M}_{\mathcal{P}})$ where $|\mathcal{P}|$ is the size of the raw point cloud and $|\mathcal{M}_{\mathcal{P}}|$ denotes the size after compression.

Another source of energy consumption is the need to sample several point clouds for each room to cover the different details. Typically, point clouds collected by normal users are not complete and have many gaps; therefore several point clouds are necessary to achieve the required modeling accuracy. Again, uploading several point clouds harms the energy budget of the participating mobile devices. To tackle such a challenge, we employ the formal grammars. Instead of relying on each point in the collected clouds, we can only extract some semantic information, such as the room width, and integrate them with the grammar to generate the required map while achieving high modeling accuracy.

## 2.3 Related Work

In this section, we discuss the research efforts in the realm of indoor mapping and crowdsensing. Specifically, we are interested in the energy-efficiency while participating in a crowdsensing system. Additionally, we review the literature to discover different methods of deriving the indoor floor plans.

### 2.3.1 Indoor Mapping

In this section, we review the main recent efforts for modeling the indoor environment. In fact, the literature has several articles tackling the problem using different resources and technologies. For instance, Philipp et al. [PBD+14] introduce MapGENIE, a new framework to derive indoor maps via collecting odometry traces. They exploited the formal grammars to enhance the model accuracy. Similarly, Alzantot et al. [AY12] present CrowdInside which is a crowdsensing-based system for the automatic construction of buildings floor plans. CrowdInside enhances the dead-reckoning accuracy through using unique anchor points which are found in typical indoor spaces for error resetting. Luo et al. [LHC+15] introduce iMap, a smartphone-based opportunistic sensing system that automatically constructs the indoor maps. iMap is designed to detect the floor plan as well as higher-level semantics such as stairs, escalators, elevators and doors. To this end, they collected several types of data including motion traces, atmospheric pressure, audio, and WiFi signal strength.

As can be seen, the aforementioned methods share collecting crowdsensed data. Although the resultant models are relatively accurate, they are limited to 2D representations. As a solution, Gao et al. [GZY+14] propose Jigsaw, a floor plan reconstruction system that leverages crowdsensed data from mobile users. Jigsaw merges images and motion traces to generate 3D point clouds of a given scene. Afterward, they compute the coordinates and orientations of objects – in the generated point clouds – to derive an initial floor plan. However, these methods require the continuous reporting of the users position and their sensed data to the corresponding serves. As an example, Jigsaw has to report around two hundred images per scene to the crowdsensing server to generate a single point cloud. This huge overhead negatively affects energy efficiency of the participating mobile devices.

We also found several articles exploiting point clouds for indoor maps. However, these point clouds are obtained using laser scanners instead of collecting them from distributed mobile devices. For example, Sanchez et al. [SZ12] observed that most building interiors can be modeled as a collection of planes representing ceilings, floors, walls and staircases. Accordingly, they use a laser scanner to collect 3D point clouds and then adopt RANSAC-based model fitting to detect ceiling and floors, as well as small-scale architectural structures, such as staircases. Similarly, Budroni et al. [BB09] introduce a modeling method based on a plane sweep algorithm for the segmentation of a point cloud in order to recognize the planar structures of a room. In this thesis, we seek to adopt point clouds as our data source while exploiting the concept of distributed crowdsensing.

As explained earlier in this chapter, Tango-powered devices enables us to acquire point clouds as well as motion traces. Although the laser scanners-generated point clouds are more accurate than those obtained by Goggle Tango devices, the level of accuracy obtained with Tango is proved to be sufficient for indoor modeling applications. Roberto et al. [RLAT16] provide an evaluation of the Tango-powered devices regarding its motion tracking and depth perception capabilities. Their results demonstrate that Tango devices sometimes present large motion tracking errors, which may harm augmented reality experience. In addition, a Tango device's depth sensing presents average error values similar to desktop depth cameras, but it is more sensitive to infrared reflection properties of the objects to be mapped. Schöps et al. [SSHP15] exploit Tango-powered devices to evaluate their real-time system for 3D reconstruction of large-scale outdoor scenes based on monocular motion stereo. Similarly, [SGSM16] use the Tango tablets to evaluate their system for automatically generating immersive and interactive virtual reality environments using the real world as a template. In this thesis, we believe that Tango-powered devices are reliable enough to be adopted for crowdsensing point clouds.

### 2.3.2 Energy Consumption

A major prerequisite for reaching user acceptance – in a typical crowdsensing system – is to acquire the necessary data in an energy-efficient manner. Specifically, energy is dissipated, due to participating in a crowdsensing system, for four main tasks, including: *data acquisition* and *uploading*, *query distribution*, and *device positioning*. Several works have been dedicated to improve the crowdsensing energy efficiency. To conserve energy due to redundant data uploading, Liu et al. [LHZ+13] propose delaying data upload until the availability of a WiFi connection in lieu of uploading the data immediately via the 3G network. The energy-efficient location API described in [DRM14] allows one to predict the location based on the assumption that most of the users repeat the same path every day.

Nevertheless, this approach may decrease the accuracy and therefore leads to significant errors in case the participating users do not follow the usual habits.

Zhao et al. [ZML14] propose a novel model of opportunistic coverage, along with a measurement methodology to estimate the coverage quality and to improve the energy efficiency. The collected mobility traces are exploited to efficiently distribute the sensing queries. Similarly, Kjaergaard et al. [KBBN11] present an energy-efficient system for tracking trajectory, in which the goal is to track straight segments of a path instead of individual coordinates. This approach is built upon the EnTracked position tracking algorithm [KLGT09], extending it to trajectory simplification and three new modes of operation: heading-aware (which employs the compass), distance-aware (which employs GPS data) and movement-aware (which employs the accelerometer). The authors show significant improvement in energy savings comparing to the EnTracked algorithm, while keeping the location error within the requested error bound.

The aforementioned approaches mainly attack services like positioning and query distribution. However, they do not consider the energy consumption due to performing the sensing tasks per se. Philipp et al. [PSA+13] propose DrOPS, a system for improving the efficiency of data acquisition in crowdsensing systems. DrOPS utilizes a model-driven approach, where the number of required readings from mobile devices is reduced by inferring readings from the model. Furthermore, the model can be used to infer readings for positions where no sensor is available. Alternatively, Ra et al. [RPKL12] propose to modify the mobile devices' hardware via the use of a dedicated low-power processor. In [LCZ+13], a piggyback mechanism is proposed that collects the data when the sensors are used by other applications. This approach significantly reduces the energy overhead, however it also degrades the number of performed measurements and does not allow for properly tracking the device mobility. Finally, Rachuri et al. [REL+14] introduce METIS, a mobile sensing platform that leverages both mobile phone sensors and fixed sensors in the environment. The system implements a novel sensing distribution scheme that is able to switch between phone and remote sensors considering the various sensing parameters, and mobility patterns of the users.

Although these methods managed to reduce the energy consumption, they are not applicable in most crowdsensing applications. For instance, predicting the readings works well with low-frequency data, such as temperature and humidity. However, it cannot be applied to other type of data, such as images and point clouds, which are necessary in our application of indoor mapping. Similarly, offloading the sensing tasks to fixed sensors requires pre-installation of such fixed sensors. In addition, adding fixed sensors to collect point clouds is not reasonable. Piggybacking the sensing queries leads to unpredictable reception time of necessary data, taking into account that most apps which use point clouds are mostly augmented reality-enhanced games. Based on these arguments, we leverage data compression and the utilization of the grammar for reducing the sensing energy overhead.

# 3 Basic Interiors Model

In this chapter we explain both the process of obtaining a floor plan from point clouds sampled with a Tango-powered device, and the concepts, algorithms and data structures each of the process steps are based on. An outline of the processing steps is shown in Figure 3.1. We will explain the function of each box in the following sections.

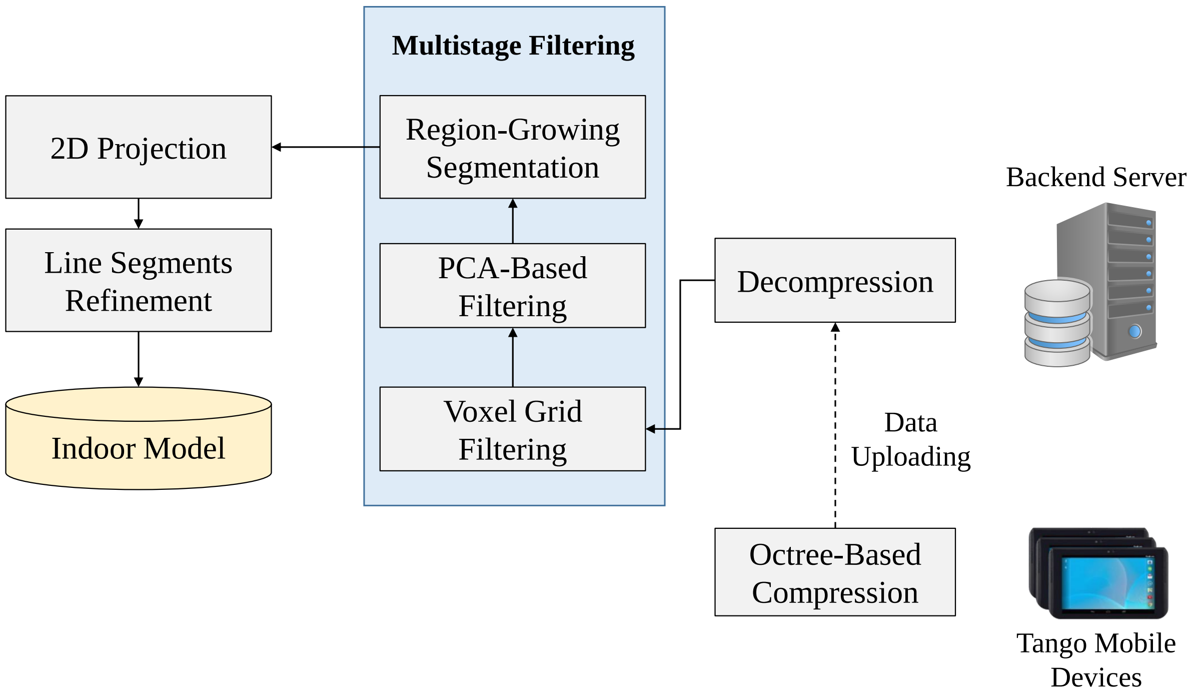


**Figure 3.1:** Processing chain in the back-end

## 3.1 Point Clouds

From a Computer Science perspective, a point cloud is a simple data structure containing a set of $n$-dimensional vectors. When these vectors are interpreted as position vectors in a Cartesian coordinate system[1], each vector represents a specific point in space. Thus, (unorganized or unstructured) point clouds describe e.g. surfaces or objects in a 3D-model, without imposing or implying any semantic relations between the contained points.

Looking at point clouds from an Object-oriented Programming (OOP) perspective, single points also can, in addition to its position, carry further information. Examples are gray-scale intensity, color information, an alpha channel or any other attribute which is required for a particular application.

### 3.1.1 Octree Compression

After a point cloud has been sampled with a Tango-powered device, octree compression is applied to save energy during data transfer to the back-end server. Octrees are a simple data structure which can be seen as an extension of binary or quad trees, where each inner node has exactly eight children (instead of two or four). They are able to handle sparse 3D data with a high memory efficiency.

Octree compression consists of three primary steps, including: a) It spatially decomposes the point cloud into an octree data structure. b) It quantizes the point cloud by replacing the points by the cell centers of the octree's leaves. c) It arithmetically encodes the remaining points by considering only cells whose child cells are nonempty. While the different parameters of octree compression are introduced in Section 5.2.2, their effects on efficiency and details about energy savings will be discussed in Section 6.1.3.

## 3.2 Voxel Rasterization

In a two-dimensional raster graphic, the smallest atomic unit is a pixel, which occupies a certain area, depending on the graphic's resolution. Adding a third dimension to a pixel results in a voxel, an element within an evenly spaced, three-dimensional grid. The term Voxel stems from the contraction of the words “volume” and “element“. Subsequently, voxel rasterization (or re-sampling) means the mapping of voxels from one grid into another with a different grid size. In almost all cases, this process entails the loss of

[1] Although Cartesian coordinate systems will be assumed throughout this thesis, other systems like polar or geo-referential systems are also conceivable in other applications.

information. For a smaller size of the new grid, he only exceptions are cases, where a) the new grid size is either smaller b) the mapping can occur without any rounding errors. If the new grid size is larger, these conditions are even more constrained. For example, a cube consisting of eight voxels can be mapped losslessly in to a grid of twice the old edge length, iff the cube aligns exactly at step boundaries of the new raster.

After the point cloud has been transferred to the server, the compressed point cloud is further processed by `pcl-model-extraction` (cf. Section 5.3.1). When the compressed point cloud is loaded, decompression and conversion to a data structure internal to the Point Cloud library is handled implicitly. Subsequently, voxel rasterization – with a larger grid size – is applied to further reduce the amount of data to be processed. This operation can be parameterized in accordance with the system resources at hand (cf. Sections 5.3.1.2 and 6.2.1).

### 3.2.1 Normals Estimation, Principal Component Analysis

For our practical purposes, we assume point clouds to be representations of surfaces, in contrast to solid, filled bodies. In order to describe the orientation of a surface (or part thereof) represented by a cluster of points within a point cloud, so-called surface normals can be used. A process of normals estimation associates each point with such a normal, and the Point Cloud Library (PCL) provides an implementation for this. Rusu provides more details on the mathematical background of his implementation in [Rus09]. In addition to a normal, each point is also attributed a curvature. The calculated surface normals and curvatures are required for both the next processing step, filtering by normals angle, and for segmentation using region growing.

### 3.2.2 Filtering by Normals Angle

Since we are primarily interested in upright walls, the number of points to process during region growing can be drastically reduced by only considering points, whose normal angle $\alpha$ against the x/y-plane lies within a certain range between $\pm\beta$. In other words, if $|\alpha| > \beta$ applies to a point's normal, the point is dismissed (cf. Sections 5.3.1.4 and 6.2.3). A perfectly sampled wall would be reflected by points with normal angles of 0°, without exception. Flat desktop surfaces, conversely, would imply normal angles of 90°.

### 3.2.3 Segmentation

Segmentation describes the process of partitioning a point cloud into smaller, distinct segments, with the points in each segment being in a semantic relation to each other. For our purpose of detecting wall segments, the semantic relation is defined singularly by the points' locations in space (and, implicitly, a sufficiently small difference of the angle between normals of neighboring points), i.e. the points of a segment are part of one, and only one, wall. Points which cannot be allocated to any segment are referred to as outliers.

#### 3.2.3.1 Region Growing

In this work, a segmentation algorithm called Region Growing is employed. Since we used the implementation provided by the PCL, this description is based on its online documentation. In particular, Algorithm 3.1 was obtained from [PCLc].

At the beginning, the point cloud's points $\{P\}$ are sorted, ascendingly by their curvatures $\{c\}$, into a set of available points $\{A\}$, which were obtained by normals estimation. Iterating over $\{A\}$ sorted that way, region growing is able to reduce the total amount of segments yielded. Now, we take the first point from $\{A\}$ and add it both to the current region $\{R_c\}$ and to a set $\{S_c\}$ called current seed points.

For each seed point $S_c^i$ we find the set $\{B_c^i\}$ of its nearest neighbors using a neighbor finding function $\Omega()$. Next, we iterate over $\{B_c^i\}$, referring to the current point thereof as $P_j$. If the angle between the normals of the current seed point $S_c$ and $P_j$ is less than the specified smoothness threshold angle $\Theta_{th}$, the point $P_j$ is added to the current region $\{R_c\}$, and then removed from the set of available points $\{A\}$. Subsequently, if the curvature value associated with $P_j$ is less than the provided threshold $c_{th}$, $P_j$ is also added to the set of seed point $\{S_c\}$ for the region currently being grown. When the set of seed points $\{S_c\}$ has become empty, the current region $\{R_c\}$ is grown completely and the next point with the smallest curvature is moved from the set of available points $\{A\}$ to a new set of current seed points $\{S_c\}$. After the set of available points has become empty, region growing has finished. All points which were not added to any region are finally added to a set called outliers.

Details on parameterization can be found in Sections 5.3.1.5 and 6.2.4.

**Algorithm 3.1** RegionGrowing

```
 1: // Input:
 2: // Point cloud: ${P}$
 3: // Point normals: ${N}$
 4: // Points' curvatures: ${c}$
 5: // Neighbor finding function: $\Omega()$
 6: // Curvature threshold: $c_{th}$
 7: // Smoothness threshold angle: $\Theta_{th}$
 8: procedure RegionGrowing(${P}$, ${N}$, ${c}$, $\Omega$, $c_{th}$, $\Theta_{th}$)
 9:     Region List $\{R\} \leftarrow \emptyset$
10:     Available points list $\{A\} \leftarrow \{1, \ldots, |P|\}$
11:     while $\{A\} \neq \emptyset$ do
12:         Current region $\{R_c\} \leftarrow \emptyset$
13:         Current seeds $\{S_c\} \leftarrow \emptyset$
14:         Point with minimum curvature in $\{A\} \rightarrow P_{min}$
15:         $\{S_c\} \leftarrow \{S_c\} \cup P_{min}$
16:         $\{R_c\} \leftarrow \{R_c\} \cup P_{min}$
17:         $\{A\} \leftarrow \{A\} \setminus P_{min}$
18:         for $i = 0$ to $|\{S_c\}|$ do
19:             Find nearest neighbors of current seed point $\{B_c^i\} \leftarrow \Omega(S_c\{i\})$
20:             for $j = 0$ to $\left|\{B_c^i\}\right|$ do
21:                 Current neighbor point $P_j \leftarrow B_c^i\{j\}$
22:                 if $P_j \in \{A\} \wedge arccos(\ \measuredangle(N\{S_c^i\},\ N\{P_j\})\ ) < \Theta_{th}$ then
23:                     $\{R_c\} \leftarrow \{R_c\} \cup P_j$
24:                     $\{A\} \leftarrow \{A\} \setminus P_j$
25:                     if $c\{P_j\} < c_{th}$ then
26:                         $\{S_c\} \leftarrow \{S_c\} \cup P_j$
27:                     end if
28:                 end if
29:             end for
30:         end for
31:         Add current region to global segment list $\{R\} \leftarrow \{R\} \cup \{R_c\}$
32:     end while
33:     return $\{R\}$
34: end procedure
```

### 3.2.4 Deriving Walls from Segments

Several further calculations have to be applied after segmentation to obtain a model reflecting walls. For brevity, the steps outlined in this section focus on one segment, while in reality each of these steps has to be performed on all segments yielded by the previous step of segmentation.

Errors in measurement at the time of sampling of point clouds cause the segments to occupy space, instead of – in case of ideal measurements – their points being aligned in a flat plane. After projecting a segment onto the x/y-plane, an oriented bounding box (OBB) is calculated. For this, Principal Component Analysis (PCA) is performed on an entire segment which yields its Eigenvectors (EVs), describing the segment's orientation. The EV's orientation to each other is rectangular, the first one being the segment's surface normal, the second and third EVs pointing in the direction of the segment's width and height. Next, the segment's centroid is calculated. The EVs and the centroid then are used to perform a rotation and translation (transformation $T$), such that the segment's EVs align with the axes of the coordinate system and the centroid is located at the origin. The four corner points of the bounding box then correspond to possible combinations of the minimum and maximum x- and y-coordinates of the segment's points: $(x_{min}/y_{min})$, $(x_{max}/y_{min})$, $(x_{min}/y_{max})$, $(x_{max}/y_{max})$. After applying the transformation $T^{-1}$ to those points, the resulting points finally describe the OBB.

Assuming that a wall is wider than it is thick, we inscribe a line segment into the OBB, such that its end points are located in the middle of the segment's lateral edges. The set of line segments derived from all OBBs then constitutes an indoor model.

### 3.2.5 Refinement

The model may, for some wall sections, contain multiple lines. An example could be a section with a large window, where the wall sections above and below the window might have been split into separate segments by region growing. It is then likely, that their ventral positions differ and thus they also yield two individual wall sections. To mitigate this, we added some steps for refinement, which are explained in detail in Section 5.3.1.6.

The model obtained in the previous step may still contain line segments for objects, which are not part of the building's structure. Filtering out such objects, like e.g. furniture, can be done in a final filtering step. For example, the cloud segment of provenance associated with a line segment would have to meet additional criteria, like e.g. minimum dimensions, in order to not get filtered out.

# 4 Grammar-Enhanced Mapping

In this chapter, we discuss the utilization of formal grammars to enhance the accuracy of the obtained model from the previous chapter. Moreover, we explain how grammar involvement leads to significantly reducing the energy consumption of the participating Tango-powered mobile devices. We start by providing a general overview of formal grammars, before we elaborate on the interiors grammar which encode structural information. Afterward, we explain the methodology by which we use the grammar as novel model fitting tool.

## 4.1 Formal Grammars

In this section, we explain the general concept behind formal grammars. Grammars are typically a set of production rules for strings in a formal language [Pow02]. The rules describe how to form strings from the language's alphabet that are valid according to the language's syntax. In other words, the rules are used for rewriting strings, along with a "start symbol" from which rewriting starts. In particular, formal grammars do not describe the meaning of the strings or what can be done with them in whatever context – only their form. In some applications, formal grammars are used as *recognizer* to determine whether a given string belongs to the language or is grammatically incorrect. Literally, *parsing* is the processing of words by breaking them down to a set of symbols and analyzing each one against the grammar of the language. Formally, a grammar $G$ is an ordered quadruple $\langle V_N, V_T, S, R\rangle$ where:

- $V_N$ is a set of grammar nonterminal symbols that can be replaced/expanded to a sequence of symbols.
- $V_T$ is a set of actual words in a language; these are the terminal symbols in a grammar that cannot be replaced by anything else, where $V_T \cap V_N = \emptyset$. The term "terminal" is supposed to conjure up the idea that it is a dead-end – no further expansion is possible.

- $S \in V_N$ is a start nonterminal symbol. All sentences are derived from $S$ by successive replacement using the production rules $R_i$ of the grammar $G$.
- $R_i$ is a set of grammar rules (aka production) that describe how to replace/exchange symbols.

The terminal symbols are typically denoted by small letters, $V_T = \{a, b, c, \cdots\}$ while the nonterminal symbols are denoted by capital letters, $V_N = \{A, B, C, \cdots\}$. To generate a string in the language using a formal grammar, we start with a string consisting of only a single *start* symbol. Then, the production rules are sequentially applied without a specific order, until a string that contains neither the start symbol nor designated *nonterminal* symbols is produced. A production rule is applied to a string by replacing one occurrence of the production rule's left-hand side in the string by that production rule's right-hand side. Any particular sequence of production rules on the start symbol yields a distinct string in the language. As an example, assume the alphabet consists of $a$ and $b$, the start symbol is $S$. The production rules are represented as follows.

$$\begin{aligned} &\text{Rule } R_1: \quad && S \rightarrow aSb \\ &\text{Rule } R_2: \quad && S \rightarrow ba \end{aligned}$$

At the outset, we start the derivation with the start symbol $S$. Subsequently, we select a rule to apply. For instance, if rule 1 is to be applied, we obtain the sequence $aSb$. Assume that rule 1 is selected again to be applied. Then, we replace $S$ with $aSb$ and therefore the output is represented by the sequence $aaSbb$. Finally, we may apply rule 2 via replacing $S$ with $ba$, then we get a new sequence $aababb$. Since the resultant sequence has neither the start symbol nor designated *nonterminal* symbols, there exist no further iterations. Accordingly, the production process has to be terminated.

In the literature, we found four main types of formal grammar, ranging from *type 0*, *type 1*, *type 2* to *type 3*. The difference between these types is that they have increasingly strict production rules and can therefore express fewer, less powerful, formal languages.

- **Unrestricted grammars (type 0).** It is the most general case in which productions are of the form $a \rightarrow b$ where both $a$ and $b$ are arbitrary strings of symbols in $V = V_T \cup V_N$, with $a$ non-null. There are no restrictions on what appears on the left or right-hand side other than the left-hand side must be non-empty.
- **Context-sensitive grammars (type 1).** Productions are of the form $\phi A \psi \rightarrow \phi \alpha \psi$ where $A \in V_N$ and $\alpha$ are arbitrary strings of symbols in $V$. Here, the symbols $\phi$ and $\psi$ provide the *context* for the production.

- **Context-free grammars (type 2).** Productions are of the form $A \to \alpha$ where $\alpha$ is an arbitrary string of symbols in $V$, and $A$ is a single nonterminal. Wherever the symbol $A$ is found, we can replace it with $\alpha$, i.e. regardless of context.
- **Regular grammars (type 3).** Productions are of the form $A \to a\alpha$, where $A \in V_T$. Specifically, the left-hand side must be a single nonterminal and the right-hand side can be either empty, a single terminal by itself or with a single nonterminal. These grammars are the most limited in terms of expressive power.

Notice that the classes form a hierarchy of increasing restrictiveness, so every type 3 production is also a type 2 production, and every type 2 production is also a type 1 production. After defining the formal grammars, we below give an overview of a context-free interiors model which encode structural information.

## 4.2 Interiors Grammars

Initially, formal interiors grammars are that kind of context-free grammars which are able to store geometric, topological and semantic information on building interiors. As aforementioned in Section 4.1, formal grammars represent object knowledge through symbols and a set of production rules. The nonterminal symbol $V_N$ which defines the starting point for all replacements is denoted by the axiom. By successively applying rules to the axiom, new sequences of symbols are generated, i.e. a room sequence.

To develop an interior model, we have to identify basic geometric primitives as well as characteristic topological properties by which an indoor model can be described [PBF13]. In general, modeling the floor's hallways is simpler than modeling rooms. The reason behind this fact lies in the linear arrangement of hallways which show a one-dimensional topology. Alternatively, the topology of room configurations on a floor is two-dimensional.

However, room arrangements are typically not created by random compositions of walls, but follow architectural principles and are subject to functional restrictions. Accordingly, the interiors grammar encodes knowledge about architectural principles and geometric restrictions. The following properties of building interiors are crucial for the interiors grammar design:

- In public buildings, each floor is usually traversed by a system of connected corridors. The intuition behind this design principle is to ensure convenient access to the rooms.

- Each floor is divided – according to the system of corridors – into hallway areas and non-hallway areas. Further, the non-hallway areas can be partitioned into smaller room units which are mostly arranged in a linear sequence parallel to the adjacent hallway.
- Depending on their function, such room units feature specific layouts. For example, in hotels or hospitals a typical room unit consists of a bedroom and a bathroom.

According to the first and second properties, two different grammars have to be used to describe hallway areas and non-hallway areas. On the one hand, hallway areas resemble a network of a linear structure. On the other hand, non-hallway areas can be derived by a spatial partitioning applied to the interspaces of the hallway areas. The third property makes it easier to model buildings where there are semantic relationships between the cascaded rooms. For instance, a secretary office, with high percentage, is located next to an executive office. In these cases, the grammar models both rooms as a single object, referred to as room unit. Below, we elaborate on the *split* grammar utilized for deriving the rooms' layout. It is important to mention, that the interiors grammar – utilized in this thesis – is not a contribution of the author. Rather, it is a tool which has been developed in the ComNSense project [Pro13].

### 4.2.1 Split Grammar

As usual, the split grammar is composed of the nonterminals $V_N$, the terminals $V_T$, the axiom $S$, and the production rules $R$. The nonterminals and terminals of the split grammar correspond to basic geometric primitives. Specifically, the set of nonterminals $V_N$ consists of the axiom $S = Space$. The symbol $Space$ represents a 2D wall that can still be decomposed in wall segments. The terminals $V_T = \{\epsilon, r_a, r_b, r_c, \cdots\}$ describe walls that are not divisible. Each terminal $r_i \in V_T$ corresponds to a class $i$ of rooms, identified by their geometric extent. It is sometimes beneficial to specify that a symbol can be replaced by nothing at all. To describe this case, the *null* symbol $\epsilon$ is utilized. Both nonterminals and terminals have attributes which determine their geometry and type. Accordingly, the production rules $R$ can be formulated as follows.

1. $R_i^{room}$: $Space \rightarrow r_i\ Space$
2. $R_n^{unit}$: $Space \rightarrow r_j \ldots r_k\ Space$ where $n \in$ room units in the building
3. $R^{\epsilon}$: $Space \rightarrow \epsilon$

The first rule inserts a room – which is identified by the terminal symbol $r_i$ – by dividing the available space into a room $r_i$ plus another $Space$. The resultant $Space$ can be further divided into rooms or room units. The second production rule inserts a room unit by also dividing the available space into a room unit and a remaining space. Finally, the third rule implies that if there is no further free space we substitute the symbol $Space$ with the null symbol $\epsilon$. This rule announces the termination of the current derivation process. Table 4.1 provides an example of a room grammar rule $R_1^{room}$ which describes a room in terms of its width, its type, and the prior probability $P(R_i)$. The latter principally stands for the relative frequency of occurrence of a room or room unit. The unit room is defined in a similar manner.

| | $R_1^{room}$ | $R_5^{unit}$ |
|---|---|---|
| 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 executive with assistant's office |

**Table 4.1:** Example for room grammar rules and room unit grammar rules

As an example, Figure 4.1 depicts one possible derivation tree (i.e. sequence of rules) to fill a non-hallway space with the split grammar. In the sequel, we explain the method by which we exploit the interiors grammar as a model fitting tool. Moreover, we clarify the advantages of adopting this novel approach in terms of the saved energy consumption as well as the improvement in the modeling accuracy.

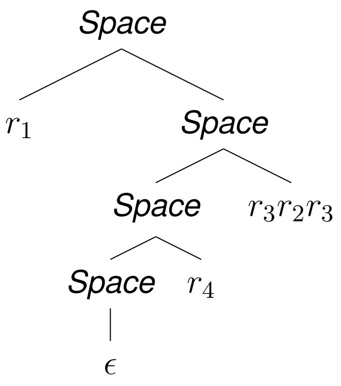


**Figure 4.1:** An example of split grammar derivation

## 4.3 Grammar-Based Model Fitting

To explain how the split grammars can be used as a model fitting tool, we have to understand the methodology by which we apply the split grammar to derive the rooms' layout. Figure 4.2 demonstrates the process of deriving the rooms' layout using the split grammar. In particular, the process of applying the production rules has to be performed on a given line, referred to as the *reference line segment* (RLS). This RLS line is used as a reference for cascading the successive rooms. The figure shows that the area between the beginning RLS_START and the end of the RLS line RLS_END, represents the free space which is then filled with rooms via applying the aforementioned production rules. The figure shows two cases of filling the RLS segment. In the first case (top RLS), the grammar inserts only rooms in accordance with the prior probability $P(R_i)$. Whereas, the split grammar – in the second case (bottom RLS) – inserts a combination of rooms and room units.

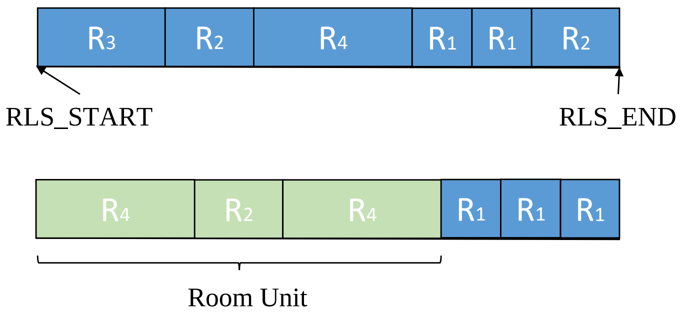


**Figure 4.2:** Rooms layout generation using a split grammar

The core idea of our proposed approach is to exploit the knowledge encoded in the grammar while processing the collected point clouds. As explained earlier, the collected point clouds are usually not complete and sometimes they do not include important details. The reason lies in collecting the point clouds on the fly, employing untrained individuals who lack the knowledge of the current processing pipeline. Figure 4.3 shows an example of such cases where the collected point cloud of a certain room is not complete. As it can be seen in the figure, several parts of the outer walls are completely missing. Therefore, it is not trivial to extract the interesting segments, i.e. walls, and then convert them into a set of interconnected walls.

To tackle this problem, we convert our strategy from relying entirely on the point clouds to enhancing the model extraction with prior knowledge embedded in the interiors grammar.

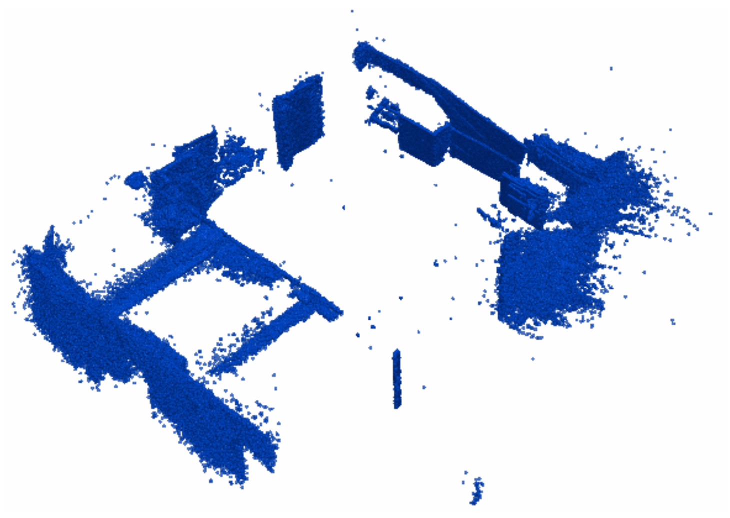


**Figure 4.3:** An example of a scanned room with missing details

Figure 4.4 depicts a modified version of the processing pipeline, which was explained the previous chapter. In this figure, we incorporate the grammar to the model extraction. We consider the basic model obtained from the previous chapter as input to the new processing steps. As Figure 4.4 demonstrates, the reference line segments are used to extract some semantic information, specifically rooms widths. To this end, we use a set of reference lines, which are provided by the indoors grammar (cf. [PBD+14]). For each RLS, we search for all line segments which are orthogonal to the current RLS and either intersect with it or have an end point located in the vicinity of the RLS. The maximum range of said vicinity is a parameter, adjustable to the model at hand. Finally, we estimate the distance between successive walls which represents the rooms widths.

Now, we have a list of room widths, obtained by processing the collected point clouds. As discussed earlier, the grammar rules have attributes such as the room width and the prior probability. In the MapGENIE system [PBD+14], the authors implemented a Markov Chain in order to randomly assign the rooms to each reference line. In our case, we convert the random walk to a deterministic one via adjusting the transition probabilities in accordance with the extracted rooms widths. Specifically, we match between the extracted room widths and the available interiors grammar to select the corresponding rules. Then, we adjust the transition probability between the successive rules according to the order, obtained from the basic model.

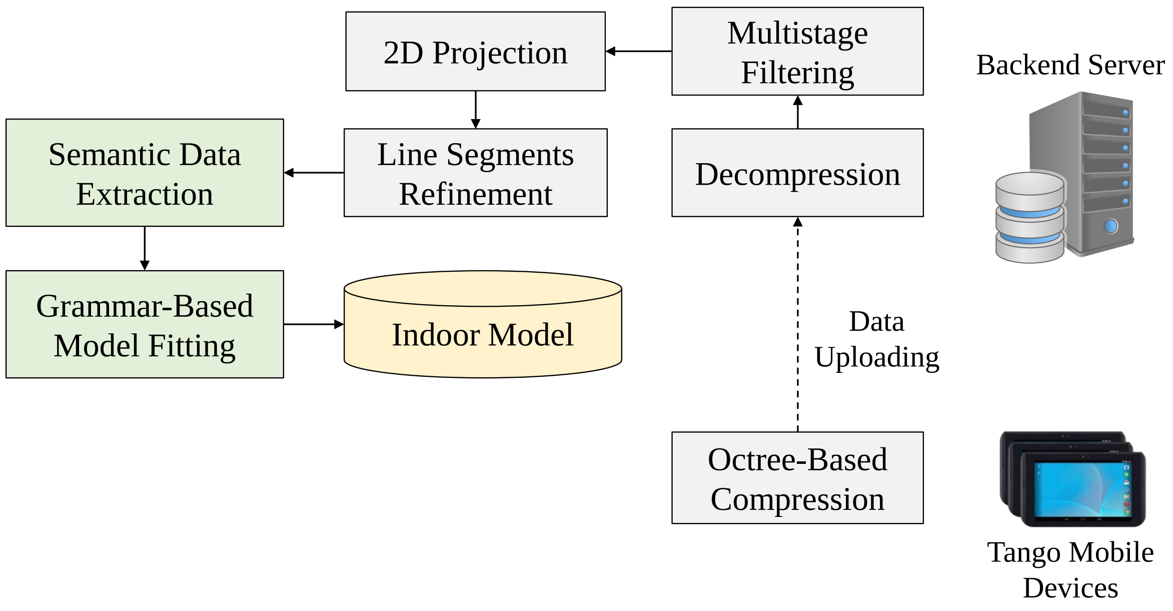


**Figure 4.4:** A modified processing chain

## 4.3.1 Discussion

The idea of incorporating the interiors grammar for enhancing the quality of the obtained basic model not only improves the modeling accuracy, but also reduces the energy consumption of the participating Tango-powered mobile devices. Instead of sending queries the users to scan several point clouds, compress them, and then upload all this data to the back-end server, we refine the system to query for only a single point cloud. Accordingly, we saved an amount of energy which had to be wasted for sampling and uploading unnecessary point clouds. In the next chapter, we discuss the implementation details of the basic model as well as the grammar-enhanced model.

# 5 Implementation

The main focus of the software that was developed in the course of this work was directed towards functionality. The proof-of-concept character of the resulting programs is reflected mainly by the fact, that they were implemented as command-line tools, rather than fully integrated GUI applications or Android apps.

All tools support several command-line parameters. Parsing is implemented using the `getopt_long()` function call (cf. [Fre]). More details about the available parameters and their meaning will be provided in the following sections introducing each tool.

CMake was chosen as a build system for the developed tools, because their main dependency, the Point Cloud Library (PCL), uses CMake as well. CMake is a cross-platform, multilingual build system, initiated back in 2000 by VTK, Inc. [Kit00].

## 5.1 Software Components

### 5.1.1 Point Cloud Library (Tango Device and Back-End)

PCL is an open-source library implemented in C++ which provides data structures and algorithms for image and point cloud processing [RC11]. It is released under the BSD license and can thereby be used freely and without charge. The software developed in the course of this thesis relies heavily on its use.

PCL data structures do neither impose nor convey any information about units associated with the coordinates of a point and only carry numerical values. In order to convey information about scale when looking at a – proportionally correct – scene, a base unit has to be assumed. Since the Tango device uses meter as a base unit for sampling point clouds, this will be assumed throughout the filtering process as well.

### 5.1.2 Point Cloud Sampling App (Tango device)

Sampling of point clouds was performed using an Android app which was developed in the course of the ComNSense project [13]. It integrates depth perception, motion tracking and area learning, as previously described in Section 2.1.1.

### 5.1.3 Application of Grammar-Based Model Fitting (Back-End)

Another component used is a tool developed in the course of the ComNSense project [13; PBD+14]. It allows for deriving indoor maps based on pedestrians' movement traces. The software was adapted to perform its calculations based on structural information extracted from sampled point clouds as opposed to its initial operating on odometric input.

## 5.2 Software implemented for the Tango Device

### 5.2.1 Point Cloud Library Cross-Compilation for Android

In order to be able to use the PCL on the Tango device, PCL and all of its dependencies have to be cross-compiled for the ARM CPU architecture and the Bionic C runtime library (Tango platform[1]). The compiler tool set used for this was the Android Native Development Kit (NDK).

The build process itself was controlled by a CMake project called pcl-superbuild, best described as an umbrella project. pcl-superbuild itself does not introduce any additional source code to be compiled. It is merely a set of definitions and instructions which automate the tasks of fetching and (cross-) compiling the source code of the PCL and all required components for Android using an NDK [Mar12b].

[1] The Android Open Source Project is not limited to the ARM architecture and the term Android platform does not necessarily imply any particular CPU architecture, although ARM is the most commonly used for Android-based devices. In contrast, the term Tango platform is intended to imply the Tango device's hardware properties and, in particular, the ARM architecture as well.

The required components comprise:

- PCL itself, [PCL11],
- Boost [Boo00], [Mar12a],
- Eigen [GJ+10],
- FLANN [ML14] and
- VTK [SML06].

pcl-superbuild defines versioned dependencies, which means that, for each dependency, a well-defined state of the dependency project's source code will be obtained for compilation (as opposed to relative version specification like, e.g. "current stable", "latest" or "master"). However the most recent activities of this project date back to the end of 2015. Thus, several adjustments to pcl-superbuild were required for it to work with versions of the listed dependencies most recent at the time of implementation of this work. These changes will now be discussed in more detail.

Except for the Eigen library, pcl-superbuild obtains all dependency projects' sources from GitHub repositories. For performance reasons, local mirrors of these repositories are highly desirable. This becomes more obvious when fully automated builds are aimed for and thus, during the development phase, numerous iterations of the entire process occur. Hence, all Git [TG05] repositories referenced by pcl-superbuild were mirrored locally on the development machine, while updating these references accordingly.

So far, pcl-superbuild used to obtain the Eigen library's source code by downloading a compressed archive via HTTP. To facilitate a homogeneous dependency management, Eigen's source code was required to be kept available in a Git repository as well. The Eigen project manages the source code in a Mercurial repository, which was easily importable into a local Git repository using git-remote-hg [Con12]. The reference in pcl-superbuild was updated to point to the Git commit corresponding with the version released at the HTTP URL.

Since PCL was going to be used on the Tango device only to perform various calculations, its visualization component was not required. The build flags of pcl-superbuild were adjusted accordingly, eliminating the dependency on VTK (and, transitively, OpenGL) all together.

The Boost libraries were pulled into the build process by boost-build [Mar12a], another CMake project. It contains both Boost's source code, initially in version 1.45.0, and the definitions required to build it using CMake. At this point, an error-free and complete build of PCL was possible.

The latest NDK version supported by pcl-superbuild was Revision 8d, using the GNU C Compiler (GCC) v4.6. Adjusting pcl-superbuild to use NDK Revision 13b revealed two problems. First, the NDK no longer shipped a file called `RELEASE.TXT` in its root directory, which was used by pcl-superbuild to detect the NDK's version and set internal build flags appropriately, depending on that version. This file was superseded by a file called `source.properties`. A quick workaround was to create the missing text file, containing just one line of text, holding the NDK's version. Second, the NDK Revision 13b provides GCC v4.9. This introduced a compilation error caused by an incompatibility with GCC newer than v4.6. More technical details on the problem and its fix can be found at [Boo11]. Subsequently, all remaining components were also updated in turn. For Eigen and FLANN, this only involved updating the Git commits referenced by pcl-superbuild.

After importing the current source code of Boost into the local Git repository and updating pcl-superbuild's reference, a new component of Boost, called `serialization`, had to be added to boost-build's CMake definitions for the build to succeed.

Since the GitHub project used for the PCL source code so far had not received any updates in several years (cf. [Mar12c]), it was replaced by a local mirror of PCL's upstream Git repository [PCL11]. Updating pcl-superbuild's reference to the new location and revision caused another compilation error. The author's fix was submitted via GitHub and got accepted by the project shortly after [Rei11]. PCL and its dependencies could then be cross-compiled for the Tango platform without build-errors.

### 5.2.2 Octree compression

For compression of point clouds, a command-line tool called pcl-pcd2octree was implemented. It reads a file (specified by `--pcd <file>`) containing a PCL point cloud data structure, whose names usually end in `.pcd`. The loaded point cloud is then compressed to a PCL Octree data structure and written to a file. All output files' names are derived from the input file and are structured as follows:

`<base name of input file>.<compression profile>.pcd-oct`[2]

PCL provides several compression profiles, which parameterize PCL's Octree compression algorithm for different usage scenarios and quality requirements. Table 5.1 shows the

[2] This extension was chosen arbitrarily and will be used throughout this work to indicate files containing an Octree data structure.

| Identifier | Profile ID |
|---|---|
| LOW_RES_OFFLINE_COMPRESSION_WITHOUT_COLOR | 1 |
| LOW_RES_OFFLINE_COMPRESSION_WITH_COLOR | - |
| MED_RES_OFFLINE_COMPRESSION_WITHOUT_COLOR | 2 |
| MED_RES_OFFLINE_COMPRESSION_WITH_COLOR | - |
| HIGH_RES_OFFLINE_COMPRESSION_WITHOUT_COLOR | 3 |
| HIGH_RES_OFFLINE_COMPRESSION_WITH_COLOR | - |
| LOW_RES_ONLINE_COMPRESSION_WITHOUT_COLOR | 4 |
| LOW_RES_ONLINE_COMPRESSION_WITH_COLOR | - |
| MED_RES_ONLINE_COMPRESSION_WITHOUT_COLOR | 5 |
| MED_RES_ONLINE_COMPRESSION_WITH_COLOR | - |
| HIGH_RES_ONLINE_COMPRESSION_WITHOUT_COLOR | 6 |
| HIGH_RES_ONLINE_COMPRESSION_WITH_COLOR | - |

**Table 5.1:** Pre-defined Octree compression profiles

internal identifiers for all pre-defined compression profiles. For brevity, henceforth the profile ID assigned therein will be used to reference a certain profile.

There are three categories of parameterization, a combination of which characterizes an Octree compression profile: quality level, processing mode and color/no color.

#### 5.2.2.1 Color Mode

Color information has no role in the assumed system model, hence models ending in "_WITH_COLOR" are not assigned an ID.

#### 5.2.2.2 Processing Mode

PCL's Octree compression algorithm comprises two modes of operation: offline and online.

In offline mode, the point cloud to be compressed is assumed to be available in its entirety at the point in time at which compression is started.

Online mode implies, that point cloud data continues to arrive as input, while the compression is being executed. This allows for encoding point cloud data (e.g. arriving from a range scanner) in near real-time, while, to some extent, preventing data-loss due to variations in system performance.

| Parameter | Compression Profile | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| point resolution [cm] | 1.00 | 0.50 | 0.01 | 1.00 | 0.50 | 0.01 |
| octree resolution [cm] | 1.00 | 0.50 | 0.01 | 1.00 | 1.00 | 1.00 |
| execute re-sampling | true | true | true | true | false | false |
| input frame rate | 100 | 100 | 100 | 50 | 40 | 30 |

**Table 5.2:** Octree compression profiles available in pcl-pcd2octree

#### 5.2.2.3 Quality Level

This parameter directly affects both the achievable compression ratio and the grade of information loss. Based on it, various parameters internal to the Octree algorithm are set: point resolution, octree resolution, execute re-sampling, input frame rate and color resolution.

Profiles eligible for use by pcl-pcd2octree and their pre-defined parameters are listed in Table 5.2.

#### 5.2.2.4 Internal Parameters

In future work in the field of this thesis, the need for a finer-grained configuration of Octree compression might arise. For that purpose, pcl-pcd2octree's source code can be modified to directly set parameters from Table 5.2 on an instantiated Object of PCL's `OctreePointCloudCompression` class. A more detailed description of these parameters is available at [PCLa].

#### 5.2.2.5 Profile Selection

The compression profile(s) to be applied when running pcl-pcd2octree can be selected by the command-line parameter `--profile-<n>` where <n> denotes a profile ID from Table 5.2.

The parameter `--all` is provided as a shortcut for specifying all profiles 1 through 6.

Specification of multiple profile parameters is supported. This is especially useful for performance evaluation, when pcl-pcd2octree is driven by a shell script to generate measurements.

The parameter `--no-output` performs compression runs on the input data using the specified profiles, however it suppresses output files from being generated. This also targets evaluation. By avoiding overhead of storage I/O, this enables measurements to be focused on CPU performance.

## 5.3 Software implemented for the Back-End Server

After pre-processing sampled point clouds on the Tango device, the compressed Octree data have to be transferred to a back-end server for model extraction. As this thesis did not focus on complete and seamless integration of all steps in the process, data transfer is yet a manual step which involves copying the relevant files from the Tango to a back-end server. In the context of crowdsensing, any method involving wireless networking is applicable for data transfer. This explicitly excludes wired transfer via an USB cable.

### 5.3.1 Model Extraction

Obtaining floor plans from sampled point clouds is the task of a command-line tool called pcl-model-extraction. It implements the filtering chain as described in Section 3, starting at the box labeled “Decompression”.

#### 5.3.1.1 File Handling

Input data on which to perform model extraction can be passed with the parameters `--octree <file>` or `--pcl <file>` which allow to read `.pcd-oct` or `.pcd` files respectively.

For the two most CPU-intensive phases, which are normals estimation and segmentation, a caching mechanism was implemented. This allows for much quicker iterations during experimental determination of an efficient set of parameters for normals estimation for, and segmentation of a given data set. Cached data comprises estimated normals for all points of the input point cloud and segments detected by the Region Growing algorithm. By default, a directory called `cache/`, relative to the specified input file, will be used. An alternate location can be set with `--cache-dir <path>`.

Normals are cached in a sub-directory thereof. Both the name of the input file and parameters pertaining to normals estimation supplied for a particular run are encoded within the name of said sub-directory. When a run with an identical set of parameters recurs, normals estimation will be replaced by reading the cached data.

Caching of detected segments works in a similar fashion, only this time the data is stored in a sub-directory of the previously created normals-cache directory. Again, the set of supplied parameters is encoded in the name of the sub-directory. On recurring runs with identical parameters, Region Growing will not be executed, and segments detected in a previous run will be loaded from cache instead. The hierarchical directory structure allows for re-use of cached normals when an identical set of parameters is supplied for normals estimation, but parameters with respect to Region Growing differ.

#### 5.3.1.2 Voxel Grid Filtering

During voxel grid filtering, a given point cloud will be re-sampled. The resolution of the resulting point cloud is adjustable separately in the direction of the z-axis and the x/y-plane. Parameters available for specifying the raster size are `--xy-distance <distance>` and `--z-distance <distance>`, respectively. To achieve the intended down-sampling to a lower resolution than provided by the input data, the raster size has to be greater than the input raster.

#### 5.3.1.3 Normals Estimation

Normals estimation depends on a radius parameter for the search of nearest-neighbors, which can be supplied by `--radius <radius>`. Aimed at batch operation, when `--normals-only` is specified, normals estimation is run, the resulting cache data are written to disk and the tool exits immediately thereafter.

#### 5.3.1.4 Filtering by Normals Angle

Specifying an angle with `--max-normals-angle <angle>` allows for filtering points by the normal angle associated with it in degrees. This parameter defines the maximum angle of a point's normal against the x/y-plane. Normals with a larger angle will be excluded from all further computations. A value of 0 means, that only points (of wall sections) which

| Parameter | Type | Default value | Special value |
|---|---|---|---|
| `--curvature <curvature>` | float | 1 | test disabled: 0.0 |
| `--min-cluster-size <size>` | positive int | 50 | - |
| `--neighbor-count <neighbors>` | positive int | 50 | - |
| `--smoothness <smoothness>` | double [°] | 3 | test disabled: 0.0 |

**Table 5.3:** Parameters pertaining to Region Growing

are perfectly perpendicular to the ground plane will be considered. Conversely, a value of 90 disables this filtering step.

#### 5.3.1.5 Segmentation

Segmentation is performed by using an algorithm called Region Growing, which is also implemented by the PCL. A detailed description thereof is provided in Section 3.2.3.1. Parameters available to influence the calculations follow the terminology used in the description. For brevity, the parameters, data types, default and special values are listed in Table 5.3.

Similar to `--normals-only`, `--region-growing-only` is provided for automation purposes to perform no further calculations after Region Growing.

Currently, for each point cloud used as input, a parameter set has to be determined empirically to yield a result suitable for further processing.

#### 5.3.1.6 Further Processing Steps

Mathematical procedures outlined in Section 3.2.4 were implemented using functionality provided by either PCL itself or by directly relying on linear algebra features provided by the Eigen library.

In order to reduce clutter, we perform a merging of segments. A merge occurs, if two segments $A$ and $B$ are oriented in the same direction (i.e. both horizontally or both vertically) and the distance $d$ of one of the points $A_1$ or $A_2$ to segment $B$ is less than a specified threshold distance $d_{th}$. The threshold can be specified with the command-line parameter `--max-merge-distance <distance>`. The actual merging consists of creating an artificial line segment $C_1C_2$, replacing the original segments $A$ and $B$. Its end points are chosen from $\{A_1, A_2\} \times \{B_1, B_2\}$ such that the distance $\overline{C_1C_2}$ is maximal.

| ID | Description |
|---:|---|
| 1 | 3D: original input cloud |
| 2 | 3D: recognized segments with outliers |
| 4 | 3D: recognized segments without outliers |
| 8 | 3D: convex hulls of segments |
| 16 | 3D: bounding boxes of segment hulls |
| 32 | 2D: recognized segments without outliers |
| 64 | 2D: convex hulls of segments |
| 128 | 2D: bounding boxes of segment hulls |
| 256 | 2D: line segments |
| 512 | 2D: line segments (without normals) |
| 1024 | 2D: line segments (without any annotations) |

**Table 5.4:** Description of available visualizations and their IDs

Since we are operating on a rectangular layout, line segments skewed by a few degrees are then aligned parallel to the coordinate system's axes. For horizontal lines, we calculate the median y-coordinate of the segment's end points and assign it to both end points. Conversely, for vertical lines the x-coordinates are replaced accordingly.

#### 5.3.1.7 Visualization

PCL provides a powerful class for visualization purposes called `PCLVisualizer`. To render a point cloud, this class has to be instantiated and a point cloud added to the created object. This displays a window with a canvas rendered by OpenGL and showing the added point cloud. Possible interactions include positional and angular adjustments of the point of view. `PCLVisualizer` provides several methods to add other primitives like e.g. line segments, polygons or text.

Visualization of the results after each step was implemented by creating a separate instance of `PCLVisualizer` and adding the respective data objects to display. Each visualization provided is assigned a numerical ID, as denoted in Table 5.4. Visualizers to be displayed can be selected with the parameter `--visualize=<selection>`. The argument `<selection>` is the sum of the IDs of visualizers to be displayed.

At this point, the implementation of extracting a floor plan, according to the basic model introduced in this work, is finished.

#### 5.3.1.8 Extraction of Grammar Rules

An enhancement of the basic model is to extract rules for grammar-based model fitting from the data model calculated thus far (cf. Section 4.3). In order to obtain a set of such rules, further processing steps, along with additional command-line parameters, are required.

To enable execution of these steps, an input file containing definitions for the reference line segments, and an output file, to which extracted rules will be written, have to be specified via `--reference-lines-file <RLS file>` and `--write-grammar-rules <rules file>`. The created output `<rules file>` can then be used to drive grammar-based model fitting as described in Section 4.3.

The expected format of the provided `<RLS file>` is a CSV text-file, containing one RLS definition per line. Each line comprises the RLS's index and the x/y-coordinates of its end points. After loading and parsing this file, for each RLS we iterate over the set of line segments from the basic model and search for branches. A branch then represents a room's wall, the distance between branches is used as the rooms' widths. In order for a line segment to be recognized as a branch of the current RLS, it must be a) oriented orthogonally to the RLS and b) be located within a distance threshold. Branches, which overshoot or intersect with a RLS are recognized without problems. However, some potential branches may not be connected to a RLS due to incomplete sampling data. The distance threshold mitigates this and can be specified via the command-line parameter `--max-branch-distance <distance>`.

After the branches for all RLSs have been determined, the results are written to the specified `<rules file>`. This also is a CSV file, containing one RLS and its branches per line:

index, Start$_x$, Start$_y$, End$_x$, End$_y$, reversed, width #1; width #2; ...; width #n;,

The column reversed contains a Boolean value, indicating whether the room widths are listed along the reference line segment in a direction leading further away from the origin (reversed = no) or towards the origin (reversed = yes).

# 6 Evaluation

Following the system model's structure, the evaluation will be split in two parts. The first part explains the system setup for sampling point clouds and data transfer to the back-end server. Extracting a floor plan on the back-end side will be the subject of the second part.

The area covered in the evaluation was the south-west quadrant on the second floor of the computer science building, where the offices of the department IPVS are located (see Figure 6.1).

## 6.1 Tango Device

### 6.1.1 Area Learning

As a first step, data required for the Tango platform to perform localization had to be acquired. The primary idea was to scan the four hallways, surrounding the target area. This would enable the Tango device to perform localization at any point in the hallway sections prior to entering a room and beginning the actual sampling of a point cloud for that particular room. Figure 6.1 shows the starting point and the initial orientation of the Tango device‘s camera when ADF data acquisition was started. This defines the origin of the global coordinate system and the orientation of its x/y-axes. The coordinate system's x-axis is directed along the left border of Figure 6.1, pointing downwards. Conversely, the y-axis runs through the depicted arrow at the origin point $(0/0)$.

The functionality for collecting ADF data was provided by the same app, that would later be used to obtain the actual point clouds for each room. In a real-world setup, in order for devices participating in point cloud sampling to have the Tango platform's localization service available, the collected ADF data would have to be distributed to all devices.

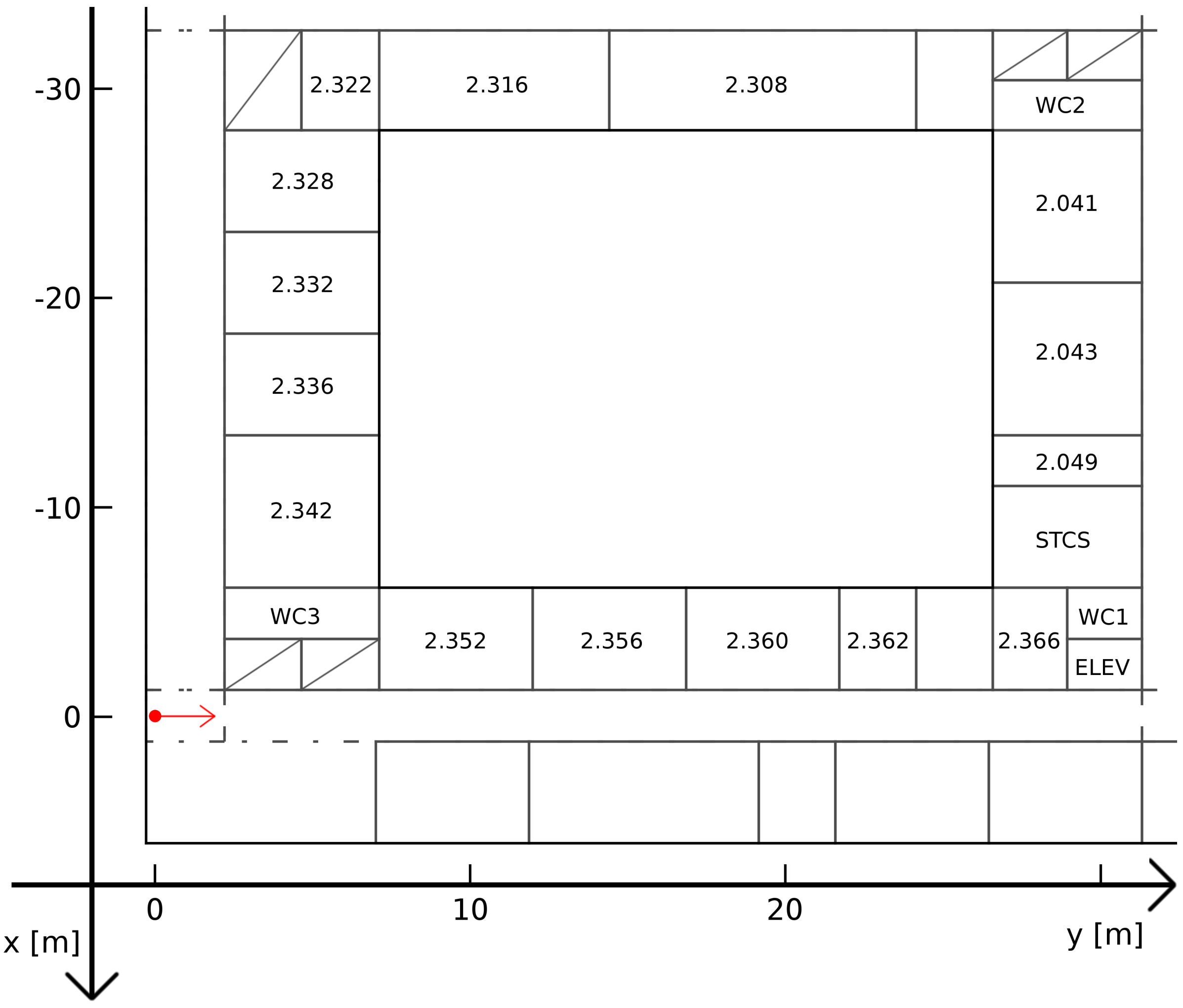


**Figure 6.1:** Ground truth of the target area

### 6.1.2 Sampling Point Clouds

After launching the sampling app, localization is performed based on the ADF data present on the device. For this, the user has to walk a few meters along the hallway, so that the Tango device can recognize distinctive markers of the surroundings and compare them to the ADF data. This usually took only a few seconds, though in areas with only less distinctive markers available, this also might take a little longer.

The point cloud sampling of all rooms of the target area was performed over the course of three days, each of which provided different lighting conditions due to sunshine or cloudiness. We noticed that both efficiency and accuracy of localization were subject to these conditions. For future experiments, as an improvement, we propose to use different

ADF data, based on the current value of the scanning device's brightness sensor. Our theory is, that the more similar the current lighting conditions are to those at the point in time when area learning was performed, the higher the efficiency and accuracy of the Tango platform's localization service should become.

After the device has located its position, the user is supposed to position herself in front of the door of the room which is about to be scanned. Scanning is initiated by the touch of a button and the user enters the room. We found it helpful, to not tilt the device too much, but to try and move the device along walls, holding it upright. Also, a minimum distance to the walls of about 1 m should be kept at all times. Scanning highly reflective surfaces, like windows or mirrors, does not yield any points for those areas at all.

After all walls are scanned, pressing the save button stops the process, and the point cloud is saved.

At this point, the sampling app does not yet provide a visual feedback about the scanned areas. We are planning to enhance the app in the near future to provide the user with such feedback, so that during scanning, a conclusion can be drawn whether a room is covered sufficiently and accurately enough.

### 6.1.3 Octree Compression

After scanning, octree compression is applied to the sampled point cloud. This minimizes the amount of data having to be transferred to the back-end server and thereby reduces the amount of energy required for submission. Our goal was, that the energy savings from transmitting compressed data significantly outweigh the additional amount of energy required to perform compression.

Using a multimeter to take voltage and current measurements directly at a device's battery of course provides the most reliable results. However we could not risk damaging the Tango device in the process. Thus we resolved to taking measurements in software. The Linux kernel running on the Tango device provides current values for battery voltage and drawn current via file entries in the `/sys` pseudo-filesystem:

- `/sys/class/power_supply/battery/current_now [µA]`
- `/sys/class/power_supply/battery/voltage_now [µV]`

During the execution of octree compression runs, these files were read every 100ms and the values, associated with timestamps, stored to log files. These measurements include the device's power consumption with the display on (at 50% brightness level),

| Room | Input File | | Output File | | Compression | | Energy |
|---|---|---|---|---|---|---|---|
| | # Points | Size [KB] | # Points | Size [KB] | Ratio [%] | Time [s] | [J] |
| FL01[1] | 152057 | 1794 | 119599 | 69 | 3.87 | 1.21 | 3.14 |
| R336 | 278286 | 3266 | 205299 | 131 | 4.02 | 1.60 | 4.56 |
| ELEV[2] | 334452 | 3900 | 125493 | 38 | 0.99 | 1.30 | 3.48 |
| FL02 | 373721 | 4329 | 224560 | 87 | 2.00 | 1.30 | 3.82 |
| WC01[3] | 380716 | 4437 | 204145 | 67 | 1.51 | 1.31 | 3.77 |
| R360 | 395085 | 4548 | 307981 | 197 | 4.33 | 1.80 | 5.75 |
| R043 | 436206 | 4949 | 352658 | 236 | 4.77 | 1.50 | 5.12 |
| R362 | 453795 | 5224 | 262432 | 115 | 2.20 | 1.20 | 3.84 |
| R041 | 467631 | 5482 | 383066 | 220 | 4.01 | 1.80 | 5.90 |
| R049 | 488919 | 5718 | 336448 | 141 | 2.46 | 1.50 | 4.83 |
| R332 | 520940 | 6083 | 391133 | 212 | 3.49 | 1.70 | 5.74 |
| R366 | 573838 | 6632 | 238737 | 80 | 1.20 | 1.50 | 4.62 |
| R308 | 607409 | 7039 | 389916 | 171 | 2.43 | 1.80 | 6.01 |
| R322 | 635939 | 7401 | 378266 | 140 | 1.89 | 1.75 | 5.69 |
| STCS[4] | 659746 | 7725 | 520927 | 318 | 4.11 | 2.30 | 8.07 |
| R316 | 716707 | 8211 | 535829 | 284 | 3.46 | 2.00 | 7.12 |
| R342 | 810227 | 9510 | 494277 | 224 | 2.36 | 2.01 | 7.10 |
| WC03 | 1187767 | 13601 | 454602 | 138 | 1.01 | 2.00 | 7.29 |
| R356 | 1450084 | 16709 | 536128 | 199 | 1.19 | 2.30 | 8.67 |
| R352 | 1588459 | 18525 | 835500 | 273 | 1.47 | 2.79 | 11.21 |
| WC02 | 1656426 | 18920 | 880432 | 317 | 1.67 | 3.00 | 11.69 |
| R328 | 1822146 | 21133 | 650803 | 189 | 0.90 | 2.80 | 10.59 |
| All merged | 15990554 | 185051 | 8819753 | 3882 | 2.10 | 23.71 | 102.83 |

**Table 6.1:** Octree compression statistics using profile #1

WiFi disabled and no other background activity than octree compression and a small overhead from reading the files in `/sys`.

Analysis of the log files yields the results depicted in Figures 6.2 and 6.3. The x-axes show point clouds per each room, ordered ascendingly, by their number of points. The final data point, 1.599m points, represents a merged point cloud, comprising the point clouds of all rooms. The values 0s and 0J (profile #3, red line) for this data point stem from the fact, that the Tango device's RAM was insufficient to support octree compression

[1]Small hallways to balcony
[2]Elevator
[3]Toilet
[4]Staircase

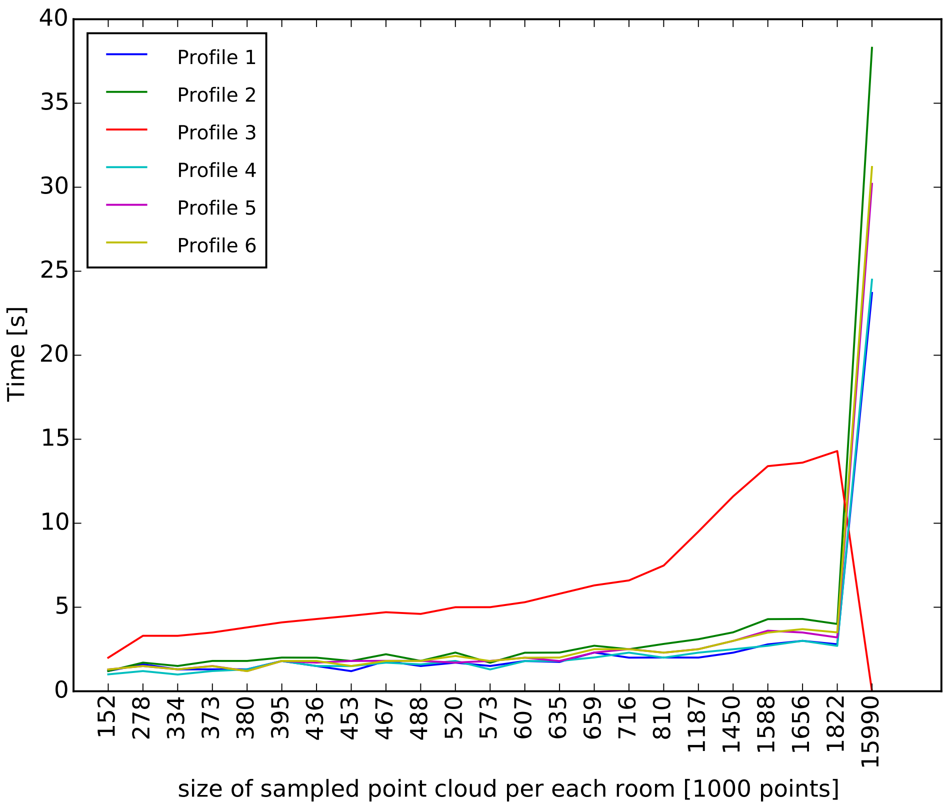


**Figure 6.2:** Octree compression – execution time

of the merged point cloud using this compression profile (cf. Table 5.1). The compression tool crashed with an out-of-memory error. The complete statistical overview for profile #1 is listed in Table 6.1.

As a result, we chose profile #1 for further experiments. Profiles ##3-6 were excluded due to the fact, that at the time octree compression starts, we assume the point cloud to be completely available in memory (cf. Section 5.2.2.2). Compared to profiles #2 and #3, profile #1 exhibits the best runtime behavior with respect to our goal of minimizing energy consumption, which in turn implies the requirement of minimal execution time.

As can be seen in in Table 6.1, the achieved compression ratios and execution times are not strictly monotonically increasing, compared to the number of points of the input clouds. This can be attributed to different noise levels present in the input clouds. Compression relies on pattern recognition and finding a more compact, algorithmic, representation for the provided input. Noise can be seen as random data, which generally is less compressible than structured data, like e.g. a cluster of points representing a wall.

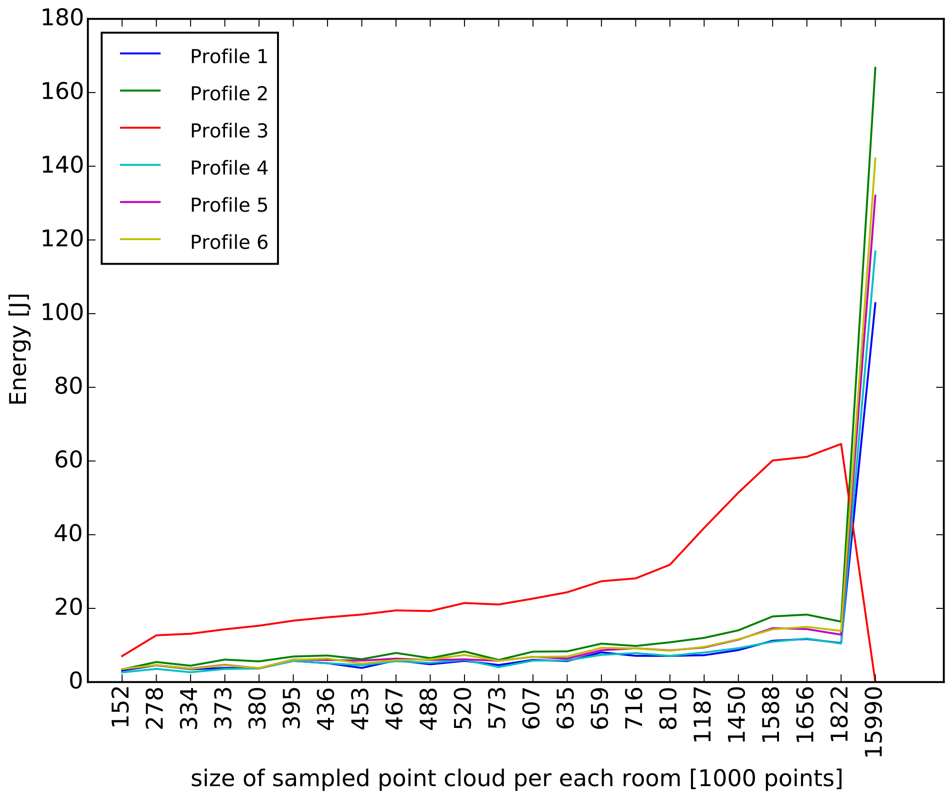


**Figure 6.3:** Octree compression – energy consumption

### 6.1.4 Energy Savings during Data Transfer

This section compares the energy consumption for uploading conventional point clouds (`.pcd` files) on the one hand, and energy required for performing octree compression and uploading octree-compressed point clouds (`.pcl-oct` files) on the other hand. The term “conventional“ point cloud is used, as opposed to “uncompressed”, since `.pcd` files contain a binary representation of point cloud data structures which is also used internally by the PCL. Saving a `.pcd` file to disk takes less time, than saving an ASCII representation of all points’ coordinates since a) less space is required and b) there is no overhead for performing a conversion of representation. To summarize, `.pcd` is the cheapest, somewhat compressed, representation of a point cloud available without any processing overhead.

During the experiment for taking measurements for the energy required for data submission, a setup similar to the one described in Section 6.1.3 was used, except for enabled WiFi networking (IEEE 802.11g). The task was to upload `.pcd` and `.pcl-oct` files for

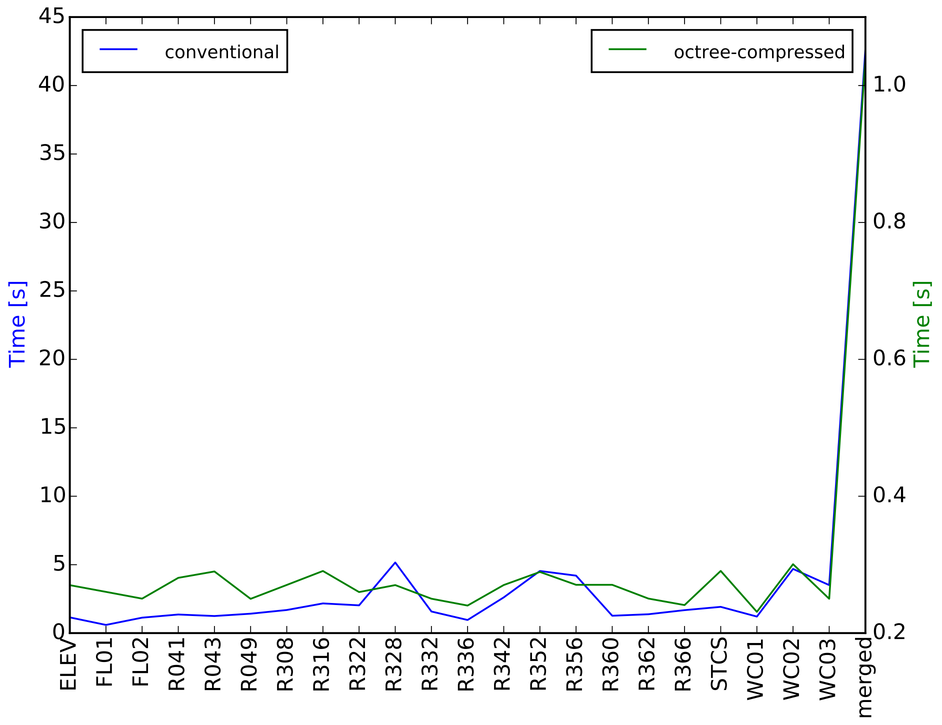


**Figure 6.4:** WiFi uploads – transfer time

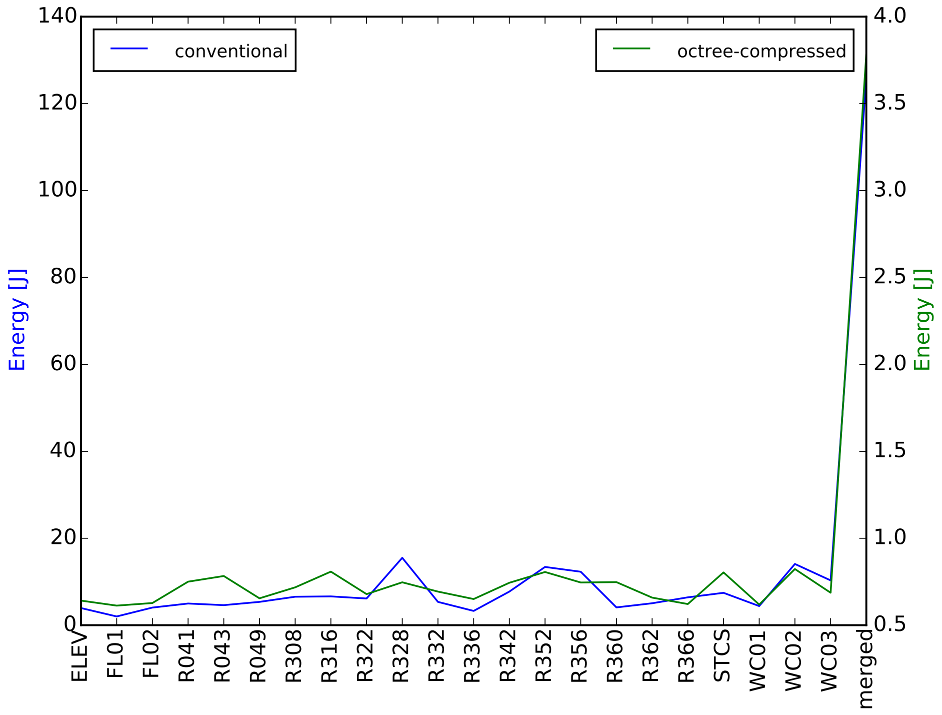


**Figure 6.5:** WiFi uploads – energy consumption

| Room | Conventional | | | Octree compressed (profile #1) | | |
|---|---|---|---|---|---|---|
| | Size [KB] | Upload Time [s] | Energy [J] | Size [KB] | Upload Time [s] | Energy [J] |
| FL01 | 1794 | 0.60 | 2.01 | 69 | 0.26 | 0.61 |
| R336 | 3266 | 0.96 | 3.28 | 131 | 0.24 | 0.65 |
| ELEV | 3900 | 1.15 | 3.94 | 38 | 0.27 | 0.64 |
| FL02 | 4329 | 1.13 | 4.05 | 87 | 0.25 | 0.63 |
| WC01 | 4437 | 1.21 | 4.40 | 67 | 0.23 | 0.62 |
| R360 | 4548 | 1.27 | 4.09 | 197 | 0.27 | 0.75 |
| R043 | 4949 | 1.25 | 4.61 | 236 | 0.29 | 0.78 |
| R362 | 5224 | 1.38 | 5.04 | 115 | 0.25 | 0.66 |
| R041 | 5482 | 1.36 | 4.98 | 220 | 0.28 | 0.75 |
| R049 | 5718 | 1.42 | 5.36 | 141 | 0.25 | 0.65 |
| R332 | 6083 | 1.58 | 5.36 | 212 | 0.25 | 0.69 |
| R366 | 6632 | 1.68 | 6.41 | 80 | 0.24 | 0.62 |
| R308 | 7039 | 1.69 | 6.55 | 171 | 0.27 | 0.72 |
| R322 | 7401 | 2.03 | 6.14 | 140 | 0.26 | 0.68 |
| STCS | 7725 | 1.92 | 7.45 | 318 | 0.29 | 0.80 |
| R316 | 8211 | 2.17 | 6.62 | 284 | 0.29 | 0.81 |
| R342 | 9510 | 2.61 | 7.73 | 224 | 0.27 | 0.74 |
| WC03 | 13601 | 3.51 | 10.30 | 138 | 0.25 | 1.69 |
| R356 | 16709 | 4.20 | 12.29 | 199 | 0.27 | 0.75 |
| R352 | 18525 | 4.54 | 13.40 | 273 | 0.29 | 0.81 |
| WC02 | 18920 | 4.69 | 14.08 | 317 | 0.30 | 0.82 |
| R328 | 21133 | 5.16 | 15.50 | 189 | 0.27 | 0.75 |
| All merged | 185051 | 42.63 | 125.99 | 3882 | 1.04 | 3.78 |

**Table 6.2:** Data upload time and energy consumption

all scanned rooms. To keep computational overhead as minimal as possible (e.g. exclude communication overhead stemming from using protocols such as HTTP or even an SSL-encrypted connection), the tool `netcat` was used to perform the actual data transfers over plain TCP connections. The receiving instance of `netcat` was running on a computer also running Linux, which was connected to the WiFi access point via a wired connection. In order to minimize fluctuations in the measurements due to jitter during radio transmissions, each file was uploaded ten times in a row, and the arithmetic mean was considered the result. Table 6.2 shows the measurements of absolute upload time and required energy, while Table 6.3 compares the efforts of conventional uploads with those of compression plus upload.

| Room | # Points | Energy [J] | | Savings | |
|---|---|---|---|---|---|
| | | Conventional Upload | Octree Compr. + Upload | Energy [J] | Percentage |
| FL01 | 152057 | 2.01 | 3.75 | -1.74 | -86.9 |
| R336 | 278286 | 3.28 | 5.21 | -1.93 | -58.7 |
| ELEV | 334452 | 3.94 | 4.12 | -0.18 | -4.6 |
| FL02 | 373721 | 4.05 | 4.45 | -0.40 | -9.8 |
| WC01 | 380716 | 4.40 | 4.39 | 0.01 | 0.2 |
| R360 | 395085 | 4.09 | 6.49 | -2.41 | -58.8 |
| R043 | 436206 | 4.61 | 5.90 | -1.29 | -28.0 |
| R362 | 453795 | 5.04 | 4.50 | 0.55 | 10.8 |
| R041 | 467631 | 4.98 | 6.65 | -1.67 | -33.5 |
| R049 | 488919 | 5.36 | 5.48 | -0.12 | -2.2 |
| R332 | 520940 | 5.36 | 6.44 | -1.08 | -20.2 |
| R366 | 573838 | 6.41 | 5.25 | 1.16 | 18.1 |
| R308 | 607409 | 6.55 | 6.73 | -0.19 | -2.8 |
| R322 | 635939 | 6.14 | 6.37 | -0.23 | -3.7 |
| STCS | 659746 | 7.45 | 8.87 | -1.43 | -19.2 |
| R316 | 716707 | 6.62 | 7.93 | -1.30 | -19.7 |
| R342 | 810227 | 7.73 | 7.84 | -0.12 | -1.5 |
| WC03 | 1187767 | 10.30 | 7.98 | 2.33 | 22.6 |
| R356 | 1450084 | 12.29 | 9.42 | 2.87 | 23.4 |
| R352 | 1588459 | 13.40 | 12.02 | 1.38 | 10.3 |
| WC02 | 1656426 | 14.08 | 12.51 | 1.57 | 11.2 |
| R328 | 1822146 | 15.50 | 11.33 | 4.17 | 26.9 |
| All merged | 15990554 | 125.99 | 106.61 | 19.37 | 15.4 |

**Table 6.3:** Energy savings by applying octree compression (profile #1)

As can be seen, for small file sizes, energy costs for compression plus upload are even higher than conventional uploads. However for larger uploads (i.e. room “WC03” and following), energy savings range between 10% and just above 25%. These results are to be expected, since every TCP connection suffers from overhead for the protocol’s three-way handshake to completely establish a connection. Hence we argue, that octree compression has the potential to save a significant amount of energy, given that further optimizations, like consolidated transfer of multiple point clouds per one TCP connection, are also implemented.

| Voxel Raster | Neighborhood Radius | Normals Estimation Execution Time |
|---:|---:|---:|
| 1cm[5] | 10cm | 80s |
| 2cm | 10cm | 9s |
| 3cm | 10cm | 3s |
| 4cm | 10cm | 2s |
| 5cm | 10cm | 1s |
| 1cm | 20cm | 331s |
| 2cm | 20cm | 32s |
| 3cm | 20cm | 8s |
| 4cm | 20cm | 3s |
| 5cm | 20cm | 1s |

**Table 6.4:** Execution times of normals estimation

## 6.2 Back-End Server – Basic Model

Once the `.pcl-oct` files of all rooms have been transferred to the back-end server, they can be assembled into a complete model of the target area. During this evaluation, we will use the octree-compressed point cloud "All merged" from Table 6.1 for further processing (cf. Figure 3.1). Unless noted otherwise, all performance metrics refer to this point cloud.

Decompression of the octree-compressed point cloud is handled implicitly by `pcl-model-extraction`, when the file is loaded from disk.

The main problem at this stage is to determine a set of parameters (cf. Section 5.3.1), both suitable for the given input data to yield a satisfactory result, and to minimize the overall execution time of the model extraction. The most expensive operations in terms of processing time are normals estimation and segmentation.

### 6.2.1 Voxel Rasterization

Using the OpenMP library, PCL supports multi-threaded normals estimation, which is a huge advantage on – today's standard – systems using multi-core CPUs and/or multiple CPUs. Table 6.4 shows some exemplary execution time statistics, pertaining to the author's development system: Intel i7-3770K@3.5GHz CPU, 4 cores (Hyper-Threading, 8 virtual cores), 16GB RAM.

While the time needed for normals estimation is acceptable in any of these cases, the execution time of region growing is much more susceptible with respect to the input point cloud's number of points. Moreover, there is no multi-threaded implementation of region growing provided by PCL, which makes it a primary concern to reduce the point count. However this stands in direct conflict with sustaining overall data quality. Therefore the voxel raster size has to be carefully weighed with respect to the point cloud to be processed an the system resources at hand. This problem could be alleviated in the future by also using a multi-threaded implementation of region growing. Parker et al. [PLL+14] not only introduced a parallel solution, but also leveraged the massive-parallel computing power of GPU cores using CUDA. PCL itself partially supports CUDA, but only few algorithms have been ported so far (cf. [PCLb]).

Rasterization is a lightweight operation. Depending on the raster size, on the author's system it took between 0.8s (5cm) and 1.7s (2cm).

For further processing, we assume a voxel raster size of 2cm. This reduced the input cloud's point count from 8.8 million to 3.6 million.

### 6.2.2 Normals Estimation

Since a single point does not "have an orientation", a point's surface normal actually represent the orientation of a surface described by its neighboring points. To this end, a neighborhood radius has to be specified. The effects of this radius on the resulting surface normals can be viewed as a low-pass (larger radius) or high-pass filter on the surface corrugations.

Our experiments suggest 0.2m to be a useful value to be used as neighborhood search radius.

### 6.2.3 Filtering by Normals Angle

As mentioned in Section 3.2.2, we use this filtering step to further reduce computation time. This parameter depends both on the quality of the input data, and on the radius used for the neighborhood search during normals estimation. If wall surfaces are reflected smoothly in the sampled point clouds, a smaller threshold can be used. Smooth surfaces

[5]As provided by the octree compression profile #1

could either be the result of scans with a low amount of flatness imperfections, or they could stem from a low-pass filtering effect caused by a relatively large radius used for normals estimation. The more pronounced any corrugations are, the larger the threshold needs to be selected, in order to not dismiss too much detail required for region growing.

In this evaluation, we used an angle of 10°.

### 6.2.4 Segmentation: Region Growing

Parameterization of region growing is yet a manual and empiric procedure, which highly depends on the input data at hand.

As observed during our experiments, larger values for the parameter neighborhood count lead to more widespread, consecutive segments, while smaller values create more boundaries and thus yield smaller segments. Also, larger values generally reduce the execution time of region growing, which might tempt to use too large values. This however leads to so-called “overgrowing” as shown in Figure 6.6, a typical weakness of the region growing algorithm. For our purposes, this manifests in region growing detecting a single segment, when we actually expected two separate segments – for two perpendicular walls – to be detected.

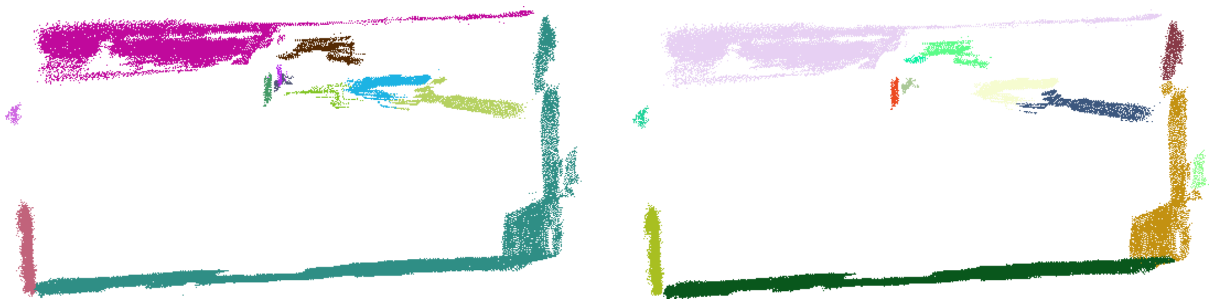


**Figure 6.6:** An example of overgrowing

The key parameter of region growing is the smoothness angle. It describes, how much disparity between the normals of the current seed point and a neighborhood point may occur, for the neighborhood point to be added to the current region. For straight wall segments, this parameter needs to be chosen relatively small. However, it is tied strongly to other parameters, especially the voxel grid size and the neighborhood radius used for normals estimation. If a coarser grid size or a smaller radius for NE is chosen, the smoothness needs to be raised, in order to make up for the details lost due to the larger grid size or the reduced low-pass filtering effect stemming from a smaller neighborhood radius. For the set of parameters mentioned so far, a smoothness angle of 1.7° yielded satisfactory results.

Initial experiments with a very coarse voxel grid size of 10cm showed, that, in addition to the smoothness angle, the curvature required tuning as well. However, we found, that for the eventually used grid size of 2cm, adjusting the curvature did not yield notable improvements. Thus, the maximum curvature was set to 1.0. This means, that, during region growing, as soon as a neighbor gets added to the current region based on normals disparity, it will also be considered as an additional seed point.

The final parameter of region growing, `--min-cluster-size <size>`, can be used to set a lower boundary for the minimum number of points a segment must contain, for it to be considered valid. Such a boundary is very useful to dismiss segments, stemming from noise or clutter, such as small, undesired objects. Note, that this value is an absolute number and therefore highly depends on the model's point density. The point density in turn depends foremost on the voxel grid size and on the amount of points filtered out by the step filtering by normals angle.

Our model was obtained using a minimum cluster size of 100 points and is shown in Figure 6.7. Outliers are shown in the left part in red.

#### 6.2.4.1 Deriving Walls from Segments and Refinement

Applying the computations described in Section 3.2.4 yields the model depicted in Figure 6.8.

Subsequently, segment merging and axes alignment are applied (cf. Section 5.3.1.6). To get rid of the many segments comprising a partial wall and replace them by a single segment, segments oriented the same way and located closer to each other than 1m are merged, which is shown in Figure 6.9.

As can be seen, this implementation's naive approach of segment merging amplifies the skew of segments. However, by exploiting the rectangular wall layout, the final processing step of axes alignment mitigates this effect. This concludes the computation of the basic model, which is depicted in Figure 6.10.

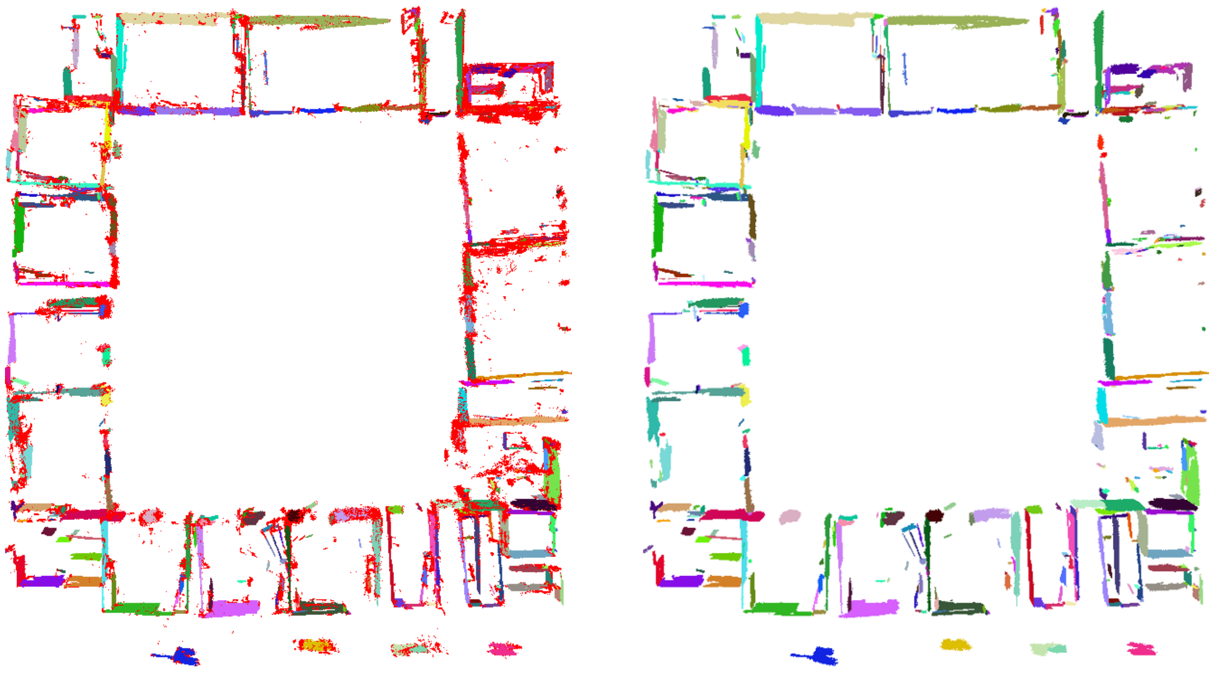


**Figure 6.7:** Segmented model of the target area (with and without outliers)

### 6.2.5 Discussion

To determine the accuracy of wall positions, we created an overlay between the result of the basic and enhanced models. Since both use different coordinate systems and origin points, axes labels were intentionally left out and the origin was defined to align with the corner of the bottom left room. In Figure 6.11, the basic model is represented by green lines, the enhanced model by blue lines. Each deviation of the basic model is annotated with the horizontal or vertical distance in centimeters. A positive value means, the basic model defines a wall location further away from the origin compared to the actual location.

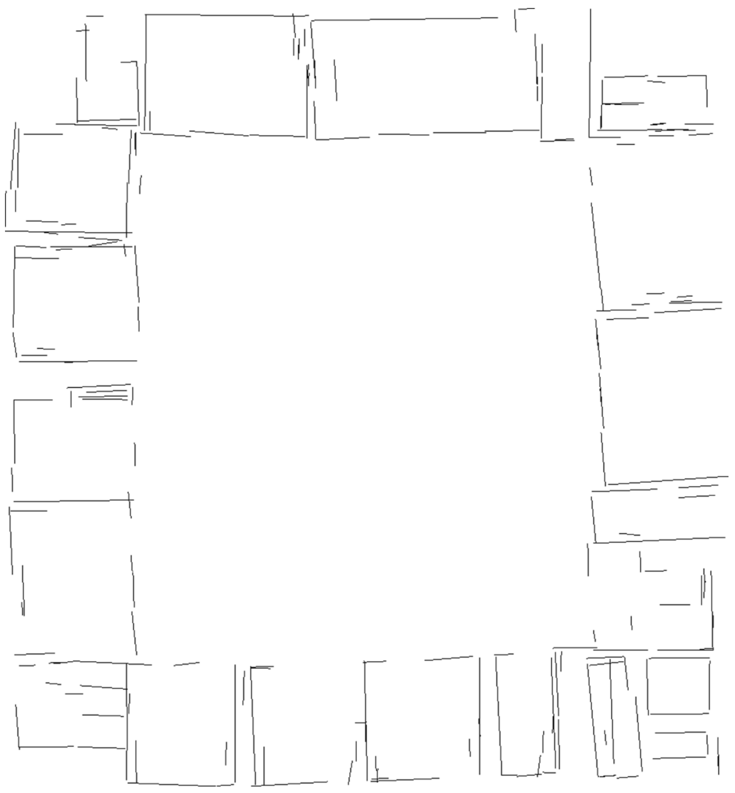


**Figure 6.8:** Model after projection and inscribing wall segments into OBBs

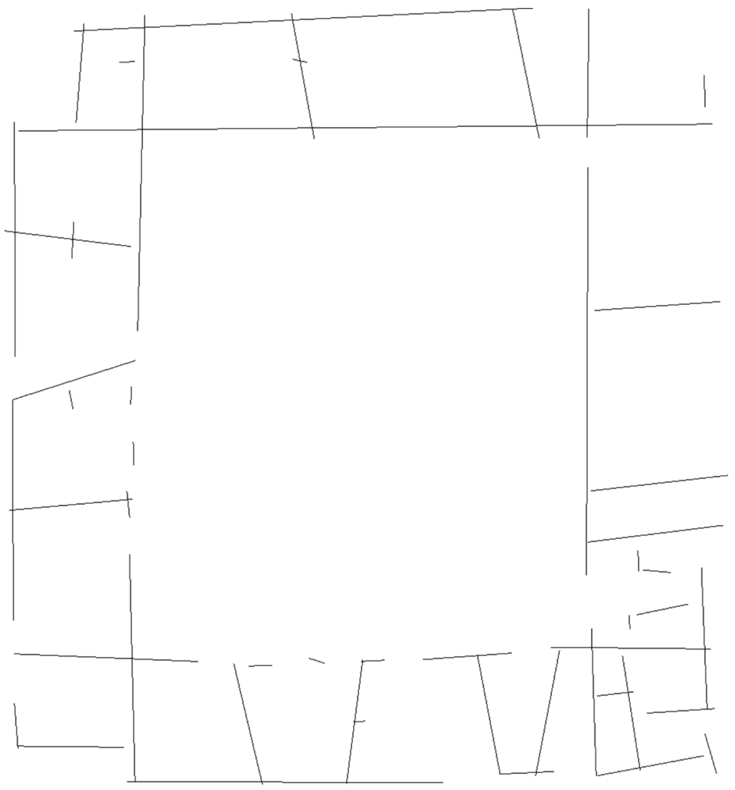


**Figure 6.9:** Model after segment merging

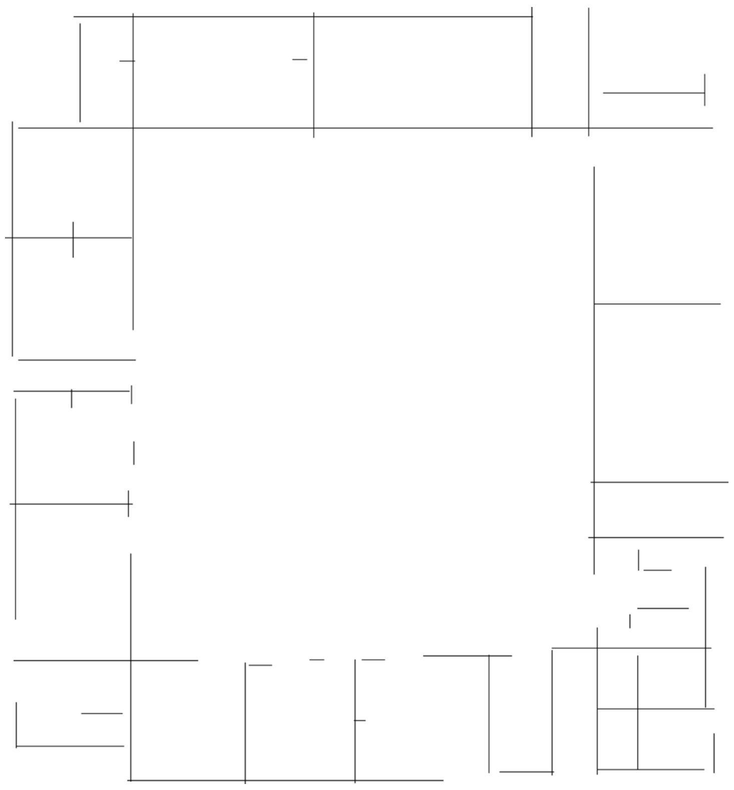


**Figure 6.10:** Model after alignment of segments with the coordinate system axes

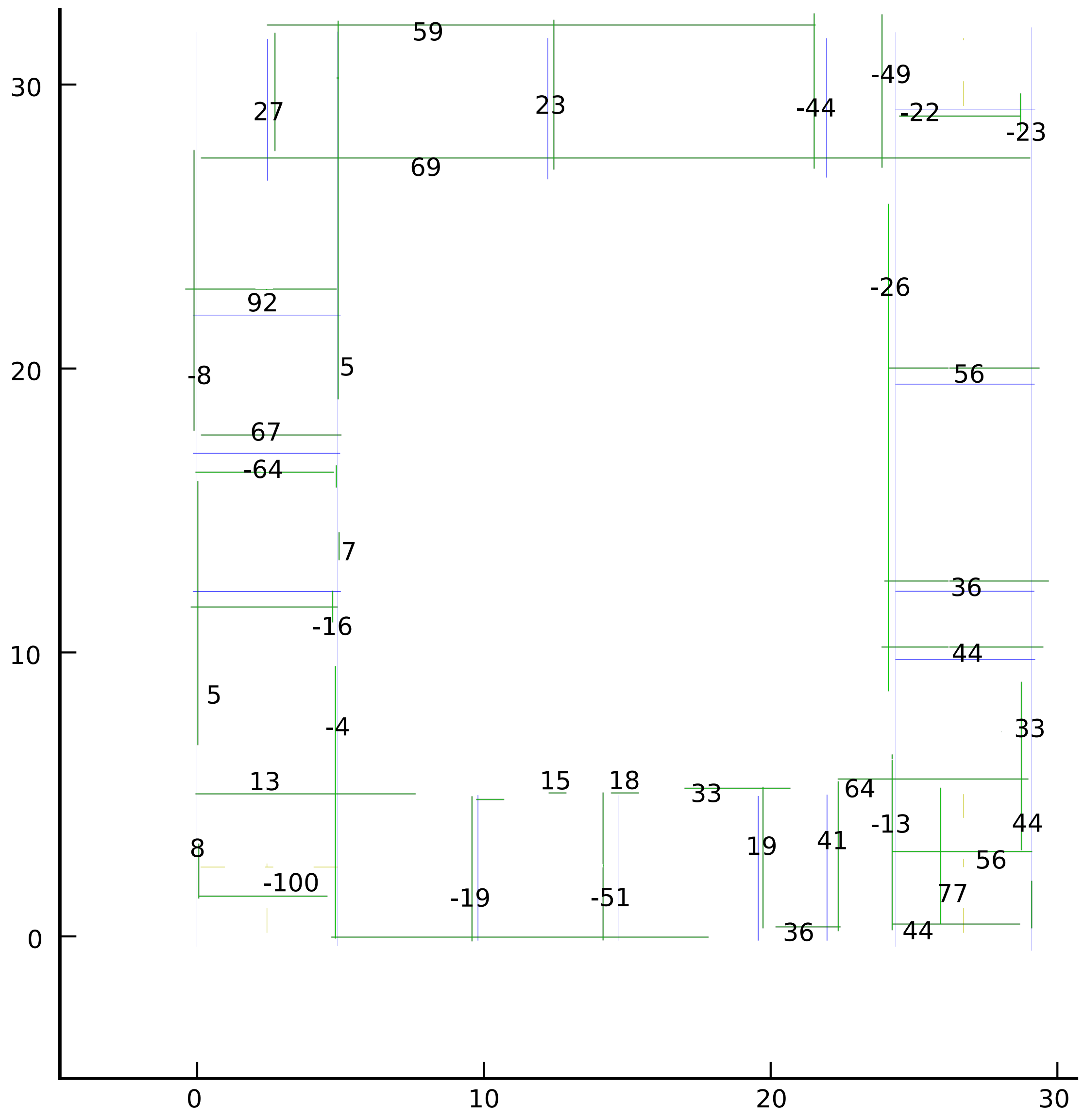


**Figure 6.11:** Deviations between the ground truth and the basic model [cm]

## 6.3 Back-End Server – Enhanced Model

Since the point clouds provided by the Tango-powered devices (cf. Figure 6.1) and the grammar-based model fitting (cf. Figure 6.12) use different coordinate systems (CS) and origins, the following conversion has to be applied (point cloud CS $\rightarrow$ grammar CS):

- apply origins offset (x: +5.55m, y: -1.65m)
- swap x- and y-axes
- flip sign of x-coordinates

To extract the room widths, we used the reference line segments as provided by the grammar (`<RLS file>, grammar CS`):

```
1,31.58711426,34.05428527,31.58711426,38.82480171
2,26.85685724,38.82480171,7.399808608,38.82480171
3,7.399808608,38.82480171,2.488114265,38.82480171
4,2.488114265,34.05428527,2.488114265,12.19798394
5,2.488114265,7.314924671,7.399808608,7.314924671
6,7.399808608,7.314924671,26.85685724,7.314924671
7,31.58711426,7.314924671,31.58711426,12.19798394
8,31.58711426,12.19798394,31.58711426,34.05428527
```

The model then generated the following `<rules file>` (grammar CS):

```
index,rlsStartX,rlsStartY,rlsEndX,rlsEndY,Reversed,rules,
1,31.587114,34.054287,31.587114,38.824802,No,0.401917;1.427078;,
2,26.856857,38.824802,7.399808,38.824802,No,0.574648;2.353300;9.059029;
   ↪ 7.482817;,
3,7.399808,38.824802,2.488114,38.824802,No,0.012743;2.193068;,
4,2.488114,34.054287,2.488114,12.197984,No,0.401915;3.765759;5.068575;1.281284;
   ↪ 4.698036;6.511435;,
5,2.488114,7.314925,7.399808,7.314925,No,4.806303;,
6,7.399808,7.314925,26.856857,7.314925,No,0.105389;4.545870;4.538753;5.570040;
   ↪ 2.603331;1.876846;,
7,31.587114,7.314925,31.587114,12.197984,No,0.488378;2.522088;2.546655;,
8,31.587114,12.197984,31.587114,34.054287,No,0.674061;4.587387;2.278649;
   ↪ 7.412082;7.306042;,
```

In addition to the input and output file parameters, we specified the threshold for maximum branch distance as 0.4m.

Using the aforementioned `<rules file>` as input for driving our indoors grammar, the model depicted in Figure 6.12 was generated.

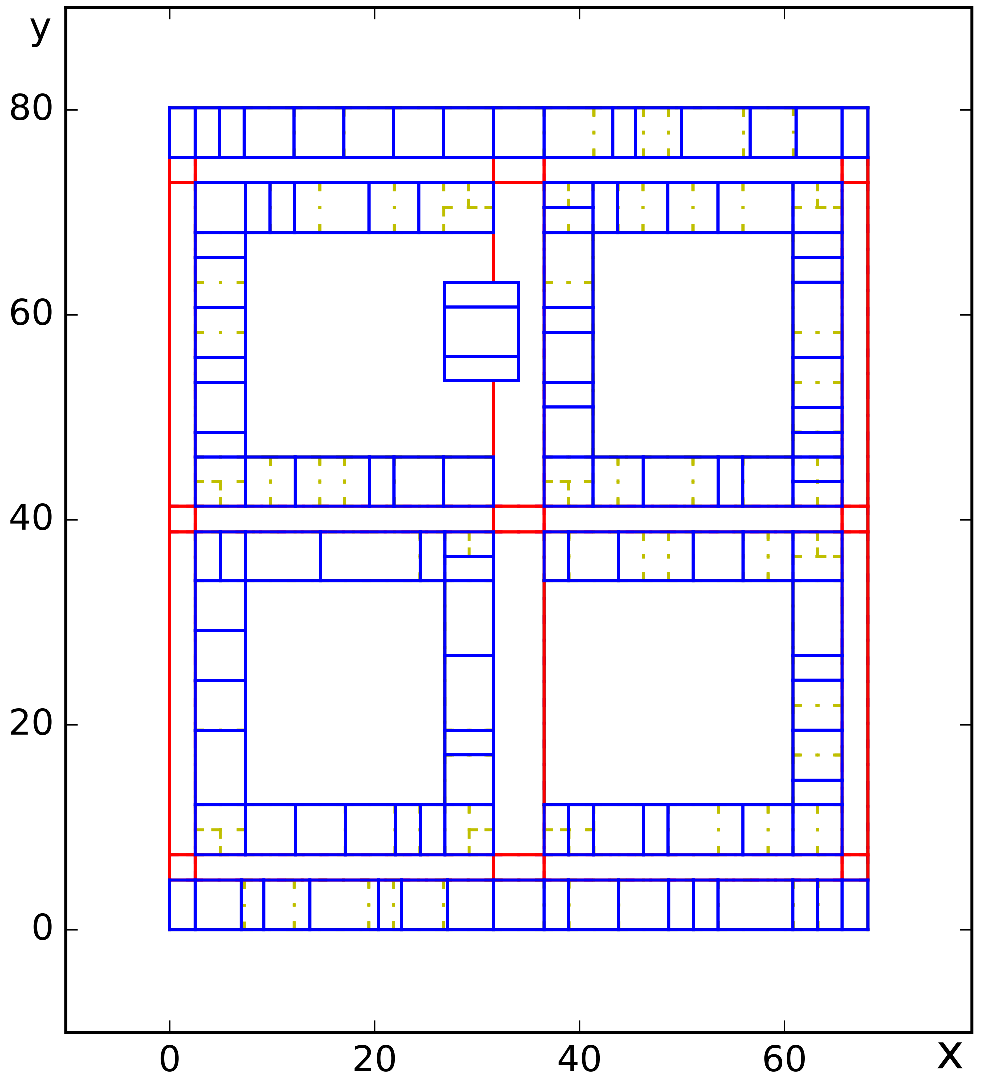


**Figure 6.12:** Grammar-enhanced model

### 6.3.1 Discussion

Table 6.5 shows reference line segments from the `<RLS file>` and the room sequences which they are supposed to create (cf. Figure 6.1).

| RLS Index | Rooms |
|---|---|
| 1 | WC2<br>Maintenance rooms |
| 2 | F2<br>2.308<br>2.316 |
| 3 | 2.322<br>Maintenance room |
| 4 | 2.328<br>2.332<br>2.336<br>2.342 |
| 5 | WC3<br>Maintenance rooms |
| 6 | 2.352<br>2.356<br>2.360<br>2.362<br>F01 |
| 7 | 2.366<br>Elevator<br>WC1 |
| 8 | Staircase<br>2.049<br>2.043<br>2.041 |

**Table 6.5:** RLSs and their associated rooms

As can be seen in the generated `<rules file>`, the first widths in rules 1-4 and 6-8 are less than 1m. In a real-world model, it does not make any sense to use such small widths for any room layout. These stem from the fact, that each first branch of those RLSs did not line up exactly with their starting point. Since the starting points are well defined by the RLSs and the grammar is robust enough in this regard, we could simply drop those values.

However, RLS #4 yielded a room width of 1.28m at the fourth position. This was caused by a slight displacement of the point cloud scanned for room 2.336 towards the origin, which is also visible in Figure 6.7. This rule had to be removed, for the grammar to produce a correct model along the left hallway section, which is shown in Figure 6.12.

The figure also shows some – manually inserted – dashed lines in the bottom left and right and the top right corners of the target area. These denote several electrical maintenance rooms, which we were not able to access. However, even if we had obtained point clouds for these rooms, the grammar still would not have produced a correct result in these areas. Currently, there is a limitation in the grammar's production rules, which prevents it from generating a layout for "cascaded" rooms. Specifically, room units, that are defined by two parallel walls branching off of a RLS, can only be split into a sequence of rooms, such that the additional wall(s) is/are parallel to the two walls defining the initial room unit.

Apart from these three limitations, the model generated for target area (bottom left quadrant) accurately reflects the ground truth with respect to room sizes and locations. The model for the remaining quadrants was continued to be generated by a random walk based on Markov Chains.

# 7 Conclusion and Future Work

## 7.1 Conclusion

In the course of this thesis, we developed a processing pipeline and the required proof-of-concept tools for obtaining indoor models from crowdsensed point clouds. To this end, we used the Google Tango-powered mobile devices to collect the point clouds in a participatory fashion. These devices provide indoors localization by integrating inertial motion tracking with area learning to reduce the localization errors. Their depth perception capability enables sampling of 3D spatial data of the device's surroundings which then can be stored as point clouds for further processing.

In crowdsensing scenarios, it is important to motivate volunteers to participate in data collection. However, energy is a scarce resource in mobile computing environments, thus energy consumption is an important factor to keep in mind when designing such scenarios. Depending on the usage pattern, besides CPU usage and display illumination, mobile communication potentially causes a high impact on overall energy consumption. Sampled point clouds have a considerable size when viewed in the context of mobile communication. To this end, we apply Octree compression, which is a highly effective method of handling sparse 3D data. This helps to significantly reduce the amount of data having to be transferred to a back-end server. In our evaluation we have shown, that the energy savings stemming from the reduced data size can outweigh the computational overhead of Octree compression.

Model extraction is a two-phase process handled by a back-end server. The first phase comprises a multi-step filtering process, followed by segmentation through region growing. Afterwards several steps of refinement are taken to reduce clutter and to closer reflect the ground truth. While the basic model may still contain many deviations and may miss several details of the target area, these shortcomings are mitigated by using the extracted information to drive a process called grammar-based model fitting. Such a formal grammar encodes prior knowledge about the layout of the modeled building in its attributed production rules.

We developed an algorithm to extract the wall segments and determine the distance between these segments, i.e. room widths. Afterward, we feed these widths to the grammar implementation – which relies on random walk on a Markov chain – so that the transition probabilities are adapted in accordance with the extracted widths. The resultant model highly outweighs the basic, i.e. grammar-free, one. The grammar not only rectified errors in room sizes, but also inserts correct rooms into areas, for which no point clouds were available due to inaccessibility. For the basic model, we had to scan the rooms several time to compensate for their incompleteness. Alternatively, utilizing the grammar for fitting the model removes the need to scan an area several times. To sum up, indoor grammars are found to be highly beneficial while deriving indoor maps from point clouds.

## 7.2 Future Work

As an outlook, the work done in this thesis can be extended in several ways, including:

- Improving the Android App so that it automatically switches between different ADF files.
- Integration of Octree compression into the android app. Due to time restrictions, we could not yet achieve the integration of the C++-based Point Cloud Library into the Java-based Android app. After this is accomplished, Octree compression could be applied in the background, possibly – depending on the devices' CPU power – even during the scanning of point clouds. Moreover, the possibility to adjust the compression parameters during a scan is required.
- Generalizing the proposed model extraction tool so that it can handle non-rectangular layouts, i.e. support curved and/or slanted walls.
- Refining the filtration stages in order to consider other geometrical semantics such as windows, doors, cabinets, tables, and chairs.

The next logical step is to undertake efforts in the generation of 3D indoors models. To that end, algorithms and computations proposed in this thesis which handle planar data need to be extended to accommodate for spatial data instead. While maintaining the idea of a two-phase process, grammar-based model fitting could also be extended such that grammars not only describe planar layouts, but also 3D interiors.

# Bibliography

[13] *The ComNSense Website*. Feb. 2013. URL: http://www.comnsense.de/ (cit. on pp. 19, 56).

[16a] *The Project Tango Website*. June 2016. URL: https://developers.google.com/tango/ (cit. on pp. 19, 28).

[16b] *The Project Tango Website - Area Learning*. June 2016. URL: https://developers.google.com/tango/overview/area-learning (cit. on p. 30).

[16c] *The Project Tango Website - Coordinate Systems*. June 2016. URL: https://developers.google.com/tango/overview/coordinate-systems (cit. on p. 31).

[16d] *The Project Tango Website - Motion Tracking*. June 2016. URL: https://developers.google.com/tango/overview/motion-tracking (cit. on p. 28).

[AAB+07] T. F. Abdelzaher, Y. Anokwa, P. Boda, J. Burke, D. Estrin, L. J. Guibas, A. Kansal, S. Madden, J. Reich. "Mobiscopes for Human Spaces." In: *IEEE Pervasive Computing* 6.2 (2007), pp. 20–29. DOI: 10.1109/MPRV.2007.38. URL: http://dx.doi.org/10.1109/MPRV.2007.38 (cit. on p. 20).

[AY12] M. Alzantot, M. Youssef. "CrowdInside: Automatic Construction of Indoor Floorplans." In: *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*. SIGSPATIAL '12. Redondo Beach, California: ACM, 2012, pp. 99–108. ISBN: 978-1-4503-1691-0. DOI: 10.1145/2424321.2424335. URL: http://doi.acm.org/10.1145/2424321.2424335 (cit. on pp. 18, 35).

[Bai15] P. Baier. "Efficient Query Distribution and Positioning in Public Sensing Systems." PhD thesis. Institute of Parallel and Distributed Systems, University of Stuttgart, 2015. DOI: 10.18419/opus-3590 (cit. on pp. 20, 21).

[BB09] A. Budroni, J. Boehm. "Toward automatic reconstruction of interiors from laser data." In: *Proceedings of Virtual Reconstruction and Visualization of Complex Architectures (3D-Arch)* (2009). URL: http://www.isprs.org/proceedings/XXXVIII/5-W1/pdf/budroni_boehm.pdf (cit. on p. 36).

[BB10] A. Budroni, J. Boehm. “Automated 3D Reconstruction of Interiors from Point Clouds.” In: *International Journal of Architectural Computing* 8.1 (2010), pp. 55–73. DOI: 10.1260/1478-0771.8.1.55. eprint: http://dx.doi.org/10.1260/1478-0771.8.1.55. URL: http://dx.doi.org/10.1260/1478-0771.8.1.55 (cit. on p. 18).

[BDR13] P. Baier, F. Durr, K. Rothermel. “Efficient distribution of sensing queries in public sensing systems.” In: *Proceedings - IEEE 10th International Conference on Mobile Ad-Hoc and Sensor Systems, MASS 2013* October (2013), pp. 272–280. DOI: 10.1109/MASS.2013.11 (cit. on p. 32).

[Boo00] Boost Project Community. *Boost C++ Libraries*. Project Website. 2000. URL: http://www.boost.org/ (cit. on p. 57).

[Boo11] Boost Project Community. *Ticket #6165*. Boost Bug Tracker. Nov. 2011. URL: https://svn.boost.org/trac/boost/ticket/6165 (cit. on p. 58).

[BPDR14] P. Baier, D. Philipp, F. Dürr, K. Rothermel. *Quality-based Adaptive Positioning for Energy-Efficient Indoor Mapping*. Englisch. Technischer Bericht Informatik 2014/06. Universität Stuttgart, Institut für Parallele und Verteilte Systeme, Verteilte Systeme: Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, Nov. 2014, p. 12. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=TR-2014-06&engl=0 (cit. on p. 17).

[CLRQ15] S. Chen, M. Li, K. Ren, C. Qiao. “Crowd Map: Accurate Reconstruction of Indoor Floor Plans from Crowdsourced Sensor-Rich Videos.” In: *2015 IEEE 35th International Conference on Distributed Computing Systems*. June 2015, pp. 1–10. DOI: 10.1109/ICDCS.2015.9 (cit. on p. 18).

[Con12] F. Contreras. *Git/Mercurial Bridge*. Project Website. Nov. 2012. URL: https://github.com/felipec/git-remote-hg (cit. on p. 57).

[DRM14] J. Duribreux, R. Rouvoy, M. Monperrus. *An Energy-efficient Location Provider for Daily Trips*. Research Report RR-8586. INRIA, Aug. 2014, p. 18. URL: https://hal.inria.fr/hal-01058830 (cit. on p. 36).

[Fre] Free Software Foundation, Inc. *The GNU C Library*. getopt_long() documentation. URL: https://www.gnu.org/software/libc/manual/html_node/Getopt.html#Getopt (cit. on p. 55).

[GJ+10] G. Guennebaud, B. Jacob, et al. *Eigen v3*. Project Website. 2010. URL: http://eigen.tuxfamily.org/ (cit. on p. 57).

[GZY+14] R. Gao, M. Zhao, T. Ye, F. Ye, Y. Wang, K. Bian, T. Wang, X. Li. “Jigsaw: Indoor Floor Plan Reconstruction via Mobile Crowdsensing.” In: *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking*. MobiCom ’14. Maui, Hawaii, USA: ACM, 2014, pp. 249–260. ISBN: 978-1-4503-2783-1. DOI: 10.1145/2639108.2639134. URL: http://doi.acm.org/10.1145/2639108.2639134 (cit. on p. 35).

[Ind16] IndoorGML. *Geometric Representation of Indoor Space*. accessed on December 2016. 2016. URL: http://docs.opengeospatial.org/is/14-005r4/14-005r4.html (cit. on p. 22).

[KBBN11] M. B. Kjærgaard, S. Bhattacharya, H. Blunck, P. Nurmi. “Energy-efficient Trajectory Tracking for Mobile Devices.” In: *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*. MobiSys ’11. Bethesda, Maryland, USA: ACM, 2011, pp. 307–320. ISBN: 978-1-4503-0643-0. DOI: 10.1145/1999995.2000025. URL: http://doi.acm.org/10.1145/1999995.2000025 (cit. on p. 37).

[Kit00] Kitware, Inc. *CMake cross-platform Build System*. Project Website. 2000. URL: https://cmake.org/ (cit. on p. 55).

[KLGT09] M. B. Kjærgaard, J. Langdal, T. Godsk, T. Toftkjær. “EnTracked: Energy-efficient Robust Position Tracking for Mobile Devices.” In: *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services*. MobiSys ’09. Kraków, Poland: ACM, 2009, pp. 221–234. ISBN: 978-1-60558-566-6. DOI: 10.1145/1555816.1555839. URL: http://doi.acm.org/10.1145/1555816.1555839 (cit. on p. 37).

[LCZ+13] N. D. Lane, Y. Chon, L. Zhou, Y. Zhang, F. Li, D. Kim, G. Ding, F. Zhao, H. Cha. “Piggyback CrowdSensing (PCS): Energy Efficient Crowdsourcing of Mobile Sensor Data by Exploiting Smartphone App Opportunities.” In: *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*. SenSys ’13. Roma, Italy: ACM, 2013, 7:1–7:14. ISBN: 978-1-4503-2027-6. DOI: 10.1145/2517351.2517372. URL: http://doi.acm.org/10.1145/2517351.2517372 (cit. on p. 37).

[LDBL07] H. Liu, H. Darabi, P. Banerjee, J. Liu. “Survey of Wireless Indoor Positioning Techniques and Systems.” In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37.6 (Nov. 2007), pp. 1067–1080. ISSN: 1094-6977. DOI: 10.1109/TSMCC.2007.905750 (cit. on p. 17).

[LHC+15] C. Luo, H. Hong, L. Cheng, K. Sankaran, M. C. Chan. "iMap: Automatic inference of indoor semantics exploiting opportunistic smartphone sensing." In: *2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. June 2015, pp. 489–497. DOI: 10.1109/SAHCN.2015.7338350 (cit. on p. 35).

[LHZ+13] H. Liu, S. Hu, W. Zheng, Z. Xie, S. Wang, P. Hui, T. Abdelzaher. "Efficient 3G budget utilization in mobile participatory sensing applications." In: *Proceedings - IEEE INFOCOM* (2013), pp. 1411–1419. ISSN: 0743166X. DOI: 10.1109/INFCOM.2013.6566935 (cit. on p. 36).

[LLY+15] D. Lymberopoulos, J. Liu, X. Yang, R. R. Choudhury, V. Handziski, S. Sen. "A Realistic Evaluation and Comparison of Indoor Location Technologies: Experiences and Lessons Learned." In: *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*. IPSN '15. Seattle, Washington: ACM, 2015, pp. 178–189. ISBN: 978-1-4503-3475-4. DOI: 10.1145/2737095.2737726. URL: http://doi.acm.org/10.1145/2737095.2737726 (cit. on p. 17).

[Mar12a] P. Marion. *Boost C++ Libraries*. GitHub Project (fork). May 2012. URL: https://github.com/patmarion/boost-build (cit. on p. 57).

[Mar12b] P. Marion. *pcl-superbuild*. GitHub Project. June 2012. URL: https://github.com/patmarion/pcl-superbuild (cit. on p. 56).

[Mar12c] P. Marion. *Point Cloud Library*. GitHub Project (fork). June 2012. URL: https://github.com/patmarion/PCL (cit. on p. 58).

[Mau12] R. Mautz. "Indoor positioning technologies." Habilitation Thesis. ETH Zurich, Department of Civil, Environmental, Geomatic Engineering, Institute of Geodesy, and Photogrammetry, 2012. DOI: http://dx.doi.org/10.3929/ethz-a-007313554. URL: https://e-collection.library.ethz.ch/view/eth:5659 (cit. on p. 17).

[Mic17] Microsoft. *Kinect for windows sensor components and specifications*. 2017. URL: https://msdn.microsoft.com/en-us/library/jj131033.aspx?f=255&MSPPError=-2147217396 (cit. on p. 28).

[ML14] M. Muja, D. G. Lowe. "Scalable Nearest Neighbor Algorithms for High Dimensional Data." In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 36 (2014). URL: https://github.com/mariusmuja/flann (cit. on p. 57).

[Ope16] OpenStreetMap. *Mapping the World via Public Sensing*. 2016. URL: https://wiki.openstreetmap.org/wiki/Main_Page (cit. on p. 22).

[PBD+14] D. Philipp, P. Baier, C. Dibak, F. Dürr, K. Rothermel, S. Becker, M. Peter, D. Fritsch. “MapGENIE: Grammar-enhanced indoor map construction from crowd-sourced data.” In: *2014 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. Mar. 2014, pp. 139–147. DOI: 10.1109/PerCom.2014.6813954. URL: ftp://ftp.informatik.uni-stuttgart.de/pub/library/ncstrl.ustuttgart_fi/INPROC-2014-03/INPROC-2014-03.pdf (cit. on pp. 18, 35, 53, 56).

[PBF13] M. Peter, S. Becker, D. Fritsch. “Grammar Supported Indoor Mapping.” In: *Proceedings of the 26th International Cartographic Conference.* (2013). URL: http://icaci.org/files/documents/ICC_proceedings/ICC2013/_extendedAbstract/283_proceeding.pdf (cit. on p. 49).

[PCLa] PCL Community. *Point Cloud Library - Compression*. Project Website. URL: http://pointclouds.org/documentation/tutorials/compression.php (cit. on p. 60).

[PCLb] PCL Community. *Point Cloud Library - CUDA support*. Project Website. URL: http://www.pointclouds.org/blog/gsoc/ioan/pcl_cuda.php (cit. on p. 77).

[PCLc] PCL Community. *Point Cloud Library - Region Growing*. Project Website. URL: http://pointclouds.org/documentation/tutorials/region_growing_segmentation.php (cit. on p. 42).

[PCL11] PCL Community. *Point Cloud Library*. GitHub Project. Mar. 2011. URL: https://github.com/PointCloudLibrary/pcl (cit. on pp. 57, 58).

[PLL+14] S. Park, J. Lee, H. Lee, J. Shin, J. Seo, K. H. Lee, Y.-G. Shin, B. Kim. “Parallelized Seeded Region Growing Using CUDA.” In: *Computational and Mathematical Methods in Medicine* 2014 (2014). DOI: 10.1155/2014/856453 (cit. on p. 77).

[Pow02] J. Power. “Notes on Formal Language Theory and Parsing.” In: *National University of Ireland, Maynooth, Kildare* (2002). URL: http://www.cs.may.ie/~jpower/Courses/parsing/parsing.pdf (cit. on p. 47).

[Pro13] T. C. Project. *Indoor Mapping*. 2013. URL: http://www.comnsense.de/mapping/index.shtml (cit. on p. 50).

[PSA+13] D. Philipp, J. Stachowiak, P. Alt, F. Dürr, K. Rothermel. "DrOPS: Model-driven optimization for Public Sensing systems." In: *2013 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. Mar. 2013, pp. 185–192. DOI: 10.1109/PerCom.2013.6526731. URL: ftp://ftp.informatik.uni-stuttgart.de/pub/library/ncstrl.ustuttgart_fi/INPROC-2013-04/INPROC-2013-04.pdf (cit. on p. 37).

[QM16] C. Qiu, M. W. Mutka. "iFrame: Dynamic indoor map construction through automatic mobile sensing." In: *2016 IEEE International Conference on Pervasive Computing and Communications (PerCom)* (2016), pp. 1–9. DOI: 10.1109/PERCOM.2016.7456500. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7456500 (cit. on p. 18).

[RC11] R. B. Rusu, S. Cousins. "3D is here: Point Cloud Library (PCL)." In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, May 2011. URL: http://pointclouds.org/ (cit. on pp. 34, 55).

[Rei11] D. Reichelt. *Pull Request #1774*. GitHub Project. Mar. 2011. URL: https://github.com/PointCloudLibrary/pcl/pull/1774 (cit. on p. 58).

[REL+14] K. K. Rachuri, C. Efstratiou, I. Leontiadis, C. Mascolo, P. J. Rentfrow. "Smartphone sensing offloading for efficiently supporting social sensing applications." In: *Pervasive and Mobile Computing* 10, Part A (2014). Selected Papers from the Eleventh Annual {IEEE} International Conference on Pervasive Computing and Communications (PerCom 2013), pp. 3–21. ISSN: 1574-1192. DOI: http://dx.doi.org/10.1016/j.pmcj.2013.10.005. URL: http://www.sciencedirect.com/science/article/pii/S1574119213001296 (cit. on p. 37).

[RLAT16] R. Roberto, J. P. Lima, T. Araújo, V. Teichrieb. "Evaluation of Motion Tracking and Depth Sensing Accuracy of the Tango Tablet." In: *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct)*. Sept. 2016, pp. 231–234. DOI: 10.1109/ISMAR-Adjunct.2016.0082 (cit. on p. 36).

[RPKL12] M.-R. Ra, B. Priyantha, A. Kansal, J. Liu. "Improving Energy Efficiency of Personal Sensing Applications with Heterogeneous Multi-processors." In: *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. UbiComp '12. Pittsburgh, Pennsylvania: ACM, 2012, pp. 1–10. ISBN: 978-1-4503-1224-0. DOI: 10.1145/2370216.2370218. URL: http://doi.acm.org/10.1145/2370216.2370218 (cit. on p. 37).

[Rus09] R. B. Rusu. “Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments.” PhD thesis. Computer Science department, Technische Universitaet Muenchen, Germany, Oct. 2009. DOI: 10.1007/978-3-642-35479-3 (cit. on p. 41).

[SGSM16] M. Sra, S. Garrido-Jurado, C. Schmandt, P. Maes. “Procedurally Generated Virtual Reality from 3D Reconstructed Physical Space.” In: *Proceedings of the 22Nd ACM Conference on Virtual Reality Software and Technology*. VRST ’16. Munich, Germany: ACM, 2016, pp. 191–200. ISBN: 978-1-4503-4491-3. DOI: 10.1145/2993369.2993372. URL: http://doi.acm.org/10.1145/2993369.2993372 (cit. on p. 36).

[SML06] W. Schroeder, K. Martin, B. Lorensen. *The Visualization Toolkit*. 4th ed. Kitware, 2006. ISBN: 9781930934191. URL: http://www.vtk.org/ (cit. on p. 57).

[SSHP15] T. Schöps, T. Sattler, C. Häne, M. Pollefeys. “3D Modeling on the Go: Interactive 3D Reconstruction of Large-Scale Scenes on Mobile Devices.” In: *2015 International Conference on 3D Vision*. Oct. 2015, pp. 291–299. DOI: 10.1109/3DV.2015.40 (cit. on p. 36).

[SZ12] V. Sanchez, A. Zakhor. “Planar 3D modeling of building interiors from point cloud data.” In: *2012 19th IEEE International Conference on Image Processing*. Sept. 2012, pp. 1777–1780. DOI: 10.1109/ICIP.2012.6467225 (cit. on p. 36).

[TG05] L. Torvalds, Git Community. *Git Version Control System*. Project Website. 2005. URL: https://www.git-scm.com/ (cit. on p. 57).

[ZML14] D. Zhao, H. Ma, L. Liu. “Energy-efficient Opportunistic Coverage for People-centric Urban Sensing.” In: *Wirel. Netw.* 20.6 (Aug. 2014), pp. 1461–1476. ISSN: 1022-0038. DOI: 10.1007/s11276-014-0687-0. URL: http://dx.doi.org/10.1007/s11276-014-0687-0 (cit. on p. 37).

All links were last followed on 2017-03-25.

**Declaration**