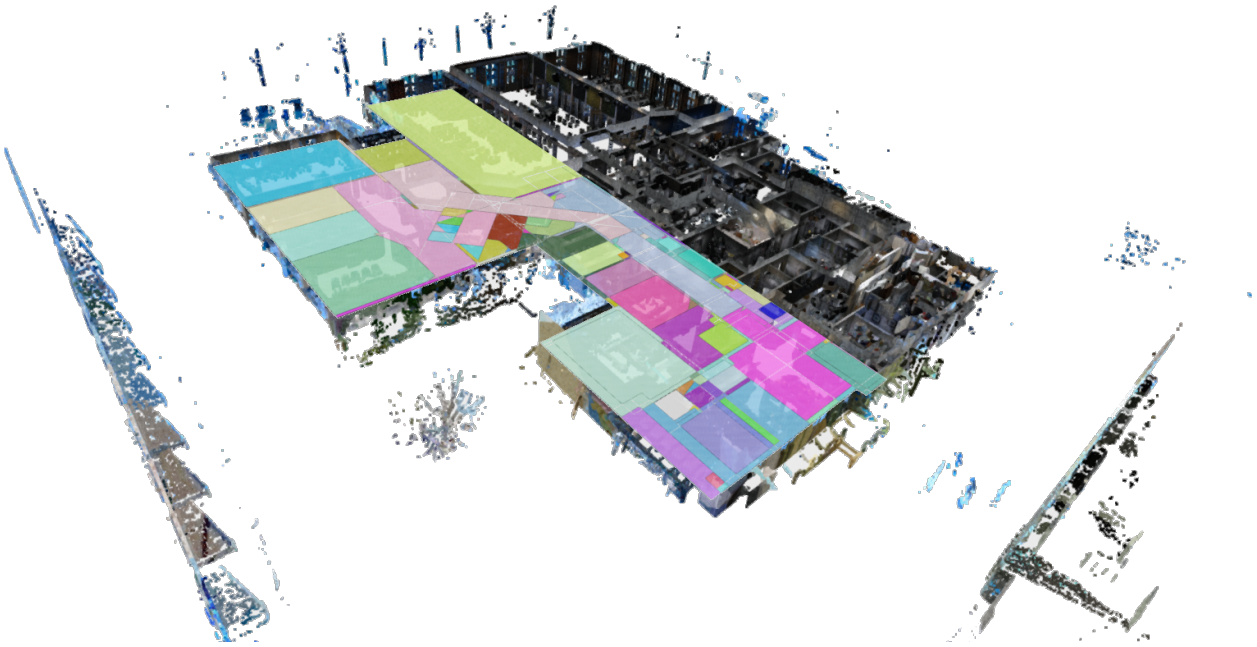
CHALMERS
UNIVERSITY OF TECHNOLOGY



Automatic Reconstruction of Indoor Spaces from 3D Point Clouds

Room level indoor plan reconstruction from cluttered LiDAR Point Clouds

Master's thesis in Complex Adaptive Systems

Eric Norman, Lovisa Landgren

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022

Automatic Reconstruction of Indoor Spaces from 3D Point Clouds

Room level indoor plan reconstruction from cluttered LiDAR Point Clouds

Eric Norman, Lovisa Landgren

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Signal processing and Biomedical engineering
Chalmers University of Technology
Gothenburg, Sweden 2022

Automatic Reconstruction of Indoor Spaces from 3D Point Clouds
Room level indoor plan reconstruction from cluttered LiDAR Point Clouds
Eric Norman, Lovisa Landgren

Supervisor: Yaroslava Lochman, Department of Electrical Engineering
Examiner: Christopher Zach, Department of Electrical Engineering

Master's Thesis 2022
Department of Electrical Engineering
Division of Signal processing and Biomedical engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Example point cloud of a scanned indoor environment, with roof and floor points removed. Some of the spaces found by the automatic model built in this thesis are overlayed in bright colors.

Typeset in LaTeX
Gothenburg, Sweden 2022

Abstract

We present a new take on the unresolved challenge of automating indoor environment reconstruction from LiDAR point clouds. Utilizing point clouds as a basis for creation of BIM models yields highly accurate results and simplifies the task substantially as compared to gathering and using manual measuring methods. It is however still a time-consuming and labor intensive process for a human to draw the model using the point cloud as a mere blueprint. We therefore attempt to automate a key part of the process end to end, namely the reconstruction of polygonal room spaces. With the goal of reaching human accuracy, if not above, we attempt to automate the steps from point cloud to a 2D room-level polygonal model of a building floor.

To this end research was conducted to combine some promising methods from different studies that have previously been done in the field into a modular data pipeline. The prototype uses multiple high-performing algorithms to denoise the point cloud, accurately identify planar room dividing components and finally define the room spaces as simple polygons.

Our work shows that end-to-end automation of room space classification is indeed possible, although lack of objective measure of room divisions poses a yet unresolved challenge. For the purpose of full BIM model reconstruction, reliable room classification is a necessity. Our work shows a promising way of combining available methods into an automatic and robust indoor environment space reconstruction process with a high level of accuracy.

Keywords: BIM, point, cloud, automatic, facility, spaces, reconstruction, polygon.

Acknowledgments

We would firstly like to thank our supervisor Yaroslava Lochman for all the support and great ideas.
Secondly, thank you to everyone at Zynka for your enthusiasm and support.

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

i, j, k	Integer indices

Sets

$\mathcal{D}$	Set of point cloud points
$\mathcal{P}$	Set of plane inlier points
$\mathcal{W}$	Set of wall patch inlier points
$\Pi_{\mathcal{W}}$	Set of planes and patch inliers
$\mathcal{S}$	Set of line segments
P	Set of polygons

Geometry

π	Plane
$r_0\left(\pi\right)$	Point on plane (center)
$\hat{n}\left(\pi\right)$	Plane normal
l	Line
$r_0\left(l\right)$	Point on line (intercept)
$T(l)$	Line tangent (normalized direction vector)
$\hat{n}\left(l\right)$	Line normal (orthogonal to tangent)
s	Line segment
a	Line segment start point

b	Line segment end point

Graphs

G	Line segment graph
C	Cell complex graph

Parameters

ξ	Voxel down sampling voxel size
ϵ	Inlier distance threshold
γ	Point cloud sample nearest neighbor radius
$\varepsilon_{\text{filter}}$	DBSCAN maximum neighbor distance for filtering
$\varepsilon_{\text{patch}}$	DBSCAN maximum neighbor distance for wall patches
$\varepsilon_{\text{diffusion}}$	DBSCAN maximum neighbor distance for room clustering
$MinPts$	DBSCAN minimum cluster points
η	Line segment padding
$h_{\angle}$	Mean shift direction bandwidth
$h_{\|\|}$	Mean shift position bandwidth
t	Diffusion time steps
σ	Diffusion weight scaling factor
$m_{\max}$	Number of eigenvectors used for diffusion embedding

Variables

A_{ij}	Diffusion graph adjacency matrix

Contents

1

Introduction

In this chapter, a problem formulation, motivation and contributions of this work are presented.

1.1 Motivation

Reconstruction of point clouds is a highly active area that has gained much progress in recent years. Creating accurate object models from point data is a complex problem that is being met in many areas at once, as automation of information processing and use of computer vision rapidly spreads in the industrial world. Active areas of research that use point cloud reconstruction involve autonomous vehicles, robotics, urban reconstruction and Building Information Modeling (BIM). Each area poses its own challenges, depending on what sort of objects that are to be classified, what level of accuracy is needed, whether the objects are static or moving, and so on. In this thesis, focus lies on reconstruction of indoor environments with focus on room layout.

The International Organization for Standardization (ISO) describe BIM as *"use of a shared digital representation of physical characteristics of a built object* [...] *to facilitate design, construction and operation processes to form a reliable basis for decisions"* [1]. A full BIM model is a refined building model that contains a high level of details, such as important objects and materials. Currently, BIM is mainly used in the construction process for new buildings. When analyzing existing facilities and developing digital models of them, a common issue is that the existing plans and blueprints are of insufficient quality, and may only exist in formats such as pdf files or paper, which are difficult to make use of.

Using point clouds as a basis when creating Building Information Models of existing facilities is far more efficient than using manual measurements of the facility spaces. The point clouds can be seen as a sample of the real spaces, that in ideal conditions contains all the information needed to extract the shapes and coordinates of the rooms. Sifting through the points to separate furniture and other objects from floor, roof and walls is often an uncomplicated task for a human. The process of manually creating the exact polygonal building models is however time consuming and labor intensive to the point that the task becomes expensive and unscalable due to the

sheer workload of labeling, editing and drawing [2]. The process is furthermore subject to human error. Automating this process would be key to facilitating the creation of highly detailed and accurate BIM models.

The problem of reconstructing 3D point clouds into polygon models is still unresolved at a large and generalized scale [3, 4, 5]. Using the information contained in the point clouds and processing it with computational methods such as clustering and shape detection algorithms has shown potential for the prospect of automating the process of creating polygonized building models in its entirety [6]. From basic use cases such as defining wall boundaries and calculating floor areas, point cloud information can also be used for identification of important building components such as pillars and doors or windows [7]. Even finer details such as air ducts, heating equipment and other functional objects may be detected from point clouds, although this depends on factors such as level of noise and what methods are used when processing the point clouds.

1.2 Problem formulation

The project is a master's thesis at Chalmers University of Technology in Gothenburg. It is carried out at Zynka BIM AB (Zynka) [8] a real estate and construction intelligence company based in Sweden. The goal of this project is to create an automatic method for reconstructing floor plans and room layouts from raw 3D point clouds. Concretely, the output of the prototype built in this project will be polygonized rooms in 2D. The room division is the sub-problem that has not yet been solved in large scale for whole buildings. Related problems such as detection of staircases and other objects in the point cloud has already been solved by others [9, 10, 7]. Therefore, an algorithm that successfully outputs the polygonized rooms could for example be combined with one that detects finer architectural details for a complete BIM output.

1.3 Our contribution

- We present a model that fully automates room-level reconstruction of Light Detection And Ranging (LiDAR) point clouds representing indoor environments.
- Our model is built as a modular pipeline, which combines some of the best point cloud reconstruction methods for high performance and adaptability to a variation of different indoor layouts.
- The model is not limited to Manhattan geometry and is robust to noise and high levels of clutter in the point clouds.
- Our Algorithm detects planar surfaces in the point cloud which are classified as candidate walls if their area is large enough.

- The candidate walls are projected onto the floor plane as a set of lines, which are refined through clustering.
- Rooms are classified in a cell complex that is built from the lines.

1.4 Prior work

A large portion of the research within the field of environment reconstruction is focused on outdoor environments, such as the outside of buildings and cities [4]. Musialski et al. propose in their survey on urban reconstruction that full automation of environment reconstruction is a challenging task due to a simultaneous context-dependency and freedom of context. They compare it to the 'chicken or egg' dilemma, as the full model needs to be derived from well-defined object primitives, while the object models need to be estimated from the raw data. They furthermore point out that the existing fully automatic algorithms rely on assumptions that are not met in practice, and that human input is generally still needed. Since a significant amount of human effort is often necessary to clean up the results of an automatic algorithm, the automation may in the end not save that much time.

The reconstruction processes of indoor and outdoor environments are similar in many ways, but the indoor environment is typically more challenging due to higher degrees of missing data and clutter, lack of coverage of all views or angles, as well as the prevalence of glass walls, windows and mirrors which cause false points and inconsistencies in the scene capture [6]. Research has been done in both areas, with focus on how methods of outdoor environment reconstruction could be adapted to work for indoor environments, and how existing indoor environment reconstruction methods could be improved.

The difference between as-designed and as-built and as-is BIMs and the importance of being able to update a model so that it corresponds well with the real world object is discussed in several research papers [3, 2]. As-designed BIMs are models created to be a blueprint for building facilities, while an as-built BIM is a model that corresponds to how the building project turned out in reality. As a facility ages it may be subject to renovation, reconstruction and removal or addition of various components. An as-is BIM is an updated model which corresponds to how the facility is in real time. Huber et al. [3] point out the usefulness of having BIMs that are up to date with changes that have been made in facilities during its lifespan, and explore available methods that could be used to automate the whole point cloud to BIM transformation process. Their algorithm however operates on single room-level, and would need to be scaled up significantly to work for whole floors or buildings.

There has been progress in reconstruction of indoor environments with many different methods, all with different advantages and disadvantages. Most of the proposed solutions use a shape detection algorithm to fit a geometrical object such as a plane [10, 9, 11] representing a wall or a box which is fitted to a wall, full room or an

entire building in the point cloud, in combination with other techniques to refine the model. An example of a method which was developed with focus on outdoor urban environment is described by Li et al. [12]. They use a box fitting algorithm to capture the general shapes of houses and buildings, that adhere to Manhattan geometry, meaning that all plane angles are either parallel or perpendicular to one another. They achieve high accuracy in their models by optimizing the distance between the points to the faces of the boxes used through use of a Markov Random Field formulation.

Sanchez et al. [9] employ Principal Component Analysis (PCA), model-fitting and RANSAC to automate planar 3D modeling of building interiors from point cloud data produced by range scanners. They identify a variety of complex objects, including staircases and interconnected walls. Their results on room-level however contain many small gaps in the walls. To create a reliable BIM model with polygonized rooms, complete room boundaries are required, and not just a large subset of the planes which make up the wall.

A recent contribution to the field of indoor environment reconstruction is presented by Yang et al. [13]. Their model reconstructs the rooms as 2D polygons, and allows for curved walls. They employ a line tracing algorithm on a horizontal slice of the point cloud. The horizontal slice is taken from above door level, where there is typically far less clutter and obstructions than on the heights closer to the floor. This method is in accordance with the concept of offset space proposed by Jung et al. [14]. The line tracing algorithm successfully traces the walls and finds rooms with a high accuracy. Creating a full 3d model with their method might however be problematic whenever high furniture is present, that reaches into the offset space.

Mura et al. [15] proposed a method for reconstruction of indoor environments, without the constraint to Manhattan geometry, which builds on cell complex generation from an arrangement of 2D lines that represents the wall planes, found using a region growing process and projected down to 2D. Adapting the method to make it work at a larger scale and for more complex rooms was deemed to be a promising staring point for this project. The algorithm however relies on the assumption that each room in the point cloud contains at least one scan point, i.e. the position of a stationary scanner. The points in the cloud are then labeled with what position they were scanned from. However, mobile scanners are becoming more and more common, and the point clouds produced by these do not contain scan points. They do contain a movement trajectory through the building which is the path taken when the area was scanned. No specific position is however associated with any certain part of the point cloud. An algorithm that functions without such additional data apart from the point cloud would therefore be useful.

While Mura et al. do not to employ RANSAC to identify planes in the point cloud, they suggest that doing so may further increase the robustness of their model. Similar to this project, they focus on outlining the rooms and more coarse architecture, and do not include detection of finer objects such as staircases, air ducts and furniture.

A couple of other studies employ RANSAC to identify planar primitives, both for outdoor and indoor environments [11, 10, 9]. Nan et al. [11] use the efficient RANSAC approach proposed by Schnabel et al. [10] to identify planar primitives from the point cloud and piece these together to get a polygonal model of an object, such as a piece of furniture or the exterior of a whole building. They refine the planar segments that are found by RANSAC to get rid of unwanted noisy elements by iteratively merging pairs of planes, and then crop the planes by the edges of the bounding box for the whole point cloud. Through pairwise intersection of the planes the shapes of the object to be classified are estimated. The planar intersections which do not correspond to any underlying object are pruned based on the relatively small amount of points in those areas. We believe that a similar approach of detecting planes, adding bounds to them and using their intersections can be done in the interior of a building to identify walls and rooms. A combination of this approach and the one proposed by Mura et al. can be an interesting starting point.

2

Background

This chapter presents a relevant background and theoretical knowledge to understand the developed method for reconstruction of indoor environments from 3D point clouds.

2.1 Point Clouds

Point clouds are sets of points in space. In this thesis, all input point clouds are created with LiDAR and Simultaneous Mapping And Localization (SLAM) in indoor environments. All point clouds mentioned in this report can therefore be assumed to be sets of 3D points. Lidar scanning is performed by targeting objects with a laser and measuring the time it takes for the beam of light to return to the receiver. This yields highly accurate depth/distance measurements. SLAM is a modern framework typically combined with LiDAR to accurately estimate the positions of a moving scanner and the geometries captured by it [6]. An example of a point clouds worked with in this project can be seen in Figure 2.1.

The point clouds at Zynka are scanned with cameras in addition to LiDAR, which yields colored point clouds. Each point is thus associated with RGB values in addition to the position coordinates. For this thesis, the color values are however not used for anything but the imagery of the report, since the company wished for a computational solution that did not rely on the color values. A reason for this is that not all rooms in the buildings that they scan can be photographed, so for some point clouds not all areas are colored.

2.1.1 Downsampling

Lidar point clouds are often very dense, making the files large and the processing slow. An issue with large point clouds is that it may be impossible to load them in the memory of a computer with regular capacity without dividing them in some way. This could be solved by loading batches of a point cloud at a time, but since there is often a higher resolution of points than needed, down sampling of point clouds is often an easier way and a typical preprocessing step.

Random down sampling is the simplest way of reducing the data and can be done

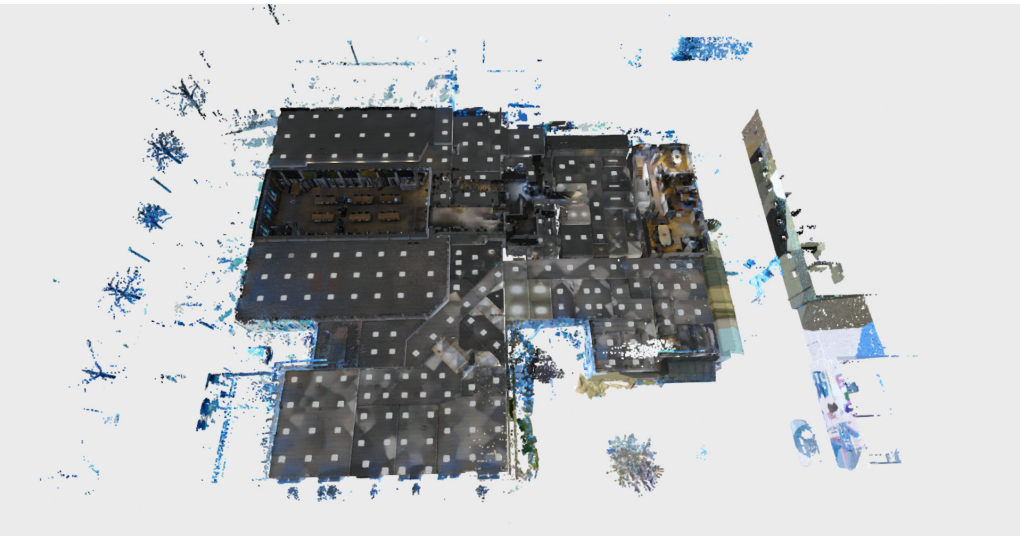


Figure 2.1: One of the point clouds from the dataset at Zynka, seen from a top-down perspective. A large amount of clutter and objects found by the LiDAR through the windows, outside the building, can be seen.

when reading the point cloud to the memory. It is done by selecting a random subset of the points in the cloud, and ignoring the rest.

Voxel downsampling is another method, that lets the density of the point cloud become uniform, as opposed to random downsampling. Voxel downsampling is done by making a grid across the whole cloud, and replacing all the points in each box of the grid with a single point computed as the mean of all ponts in the box.

2.1.2 Outlier removal

Outliers are points that do not conform to an underlying shape in the point cloud. These can be caused by noise or mirroring effects of windows, glass walls and mirrors, to name a few. To simplify processing and shape detection, removing as many outliers as possible is often advantageous.

A straightforward method for removing outliers is *radius outlier removal* where the number of neighbors of each point within a sphere of a given radius is calculated [16]. If this number is below another threshold, the point is considered an outlier and is removed.

Another approach is called *statistical outlier removal* [16]. An average distance to the n nearest neighbors is calculated and a threshold is used to remove each point that is too far away from its neighbors.

2.2 Robust Shape Detection

RANdom SAmple Consensus (RANSAC) [17] is a general iterative algorithm first proposed by Fischler and Bolles in 1981. It is used to find the parameters for mathematical models in a data set $\boldsymbol{D}$ which contains outliers to the model that do not affect the model itself. As such it can also be used to detect and prune the outliers. The algorithm works by taking random samples of data-points to identify whether they conform to a given mathematical model. The number of sampled points is the minimum amount required to solve the equation for the mathematical model. If the model describes a line for instance, two points are needed. The random sample is used to hypothesize the model. All data-points in the set are then evaluated against the hypothesized model, and RANSAC selects the best model in terms of the evaluation. In Algorithm 6 the steps for RANSAC are described in pseudo-code, much like the description by Derpanis [18].

Algorithm 1: RANSAC

1. Make a random selection of the minimum number of data points needed to solve the equation for the model parameters.
2. Solve the equation for the model parameters.
3. Determine how many points out of all data points in the set that fit into the model, within a predefined tolerance ϵ.
4. If the model is better (i.e. has more inliers) than the previous best model, save the model as the best model.
5. Repeat steps 1 through 5 N times.
6. Re-estimate the best model parameters using all of its inliers and return it.

2.3 Clustering Algorithms

Clustering algorithms divide sets of data points into clusters based on one or several defining features of the data. They are highly useful for processing of point clouds, to find distinct objects and shapes. The defining features when clustering point clouds are typically spatial proximity, density, or color.

2.3.1 DBSCAN

One useful clustering algorithm is *DBSCAN* which detects clusters of arbitrary shape in spatial data, such as point clouds. It is density based, meaning that it favours dense regions when forming clusters. Data points will likely belong to the same cluster not only if they are spatially close enough, but also if they have enough neighbors in-between such that they are both part of a dense group of points [19].

The ϵ-neighborhood of a point is defined by the following:

$$N(\epsilon, p) = \{q \mid q \in D \wedge \operatorname{dist}(p, q) \leq \epsilon\}. \tag{2.1}$$

Here, $N(\epsilon, p)$ is the neighborhood of the point p with distance threshold ϵ. q is any point in the set D of points. A point can be directly density-reachable if the following condition is met

$$\text{DDR}\,(\epsilon, p) = (p \in N(\epsilon, q)) \wedge |N(\epsilon, q)| \geq MinPts, \tag{2.2}$$

this means that a point p is in the neighborhood of q, and that the neighborhood has a number $MinPts$ of points or more. The clusters are found by sampling points and finding other points that are density reachable from this point. Clusters with less than $MinPts$ points are considered to be noise.

2.3.2 Mean Shift

Mean shift is an iterative clustering algorithm that shifts the data points towards the mode (point of maximum density) of the data and assigns them to the corresponding clusters [20]. To shift the points towards the mode we need a probability density distribution over the dataset, this can be constructed using a *kernel* that is a weighting function that is applied on the data and summed to give a probability density distribution.

The two most common kernels used for mean shift is the *flat kernel* (Eq 2.3) and the *Gaussian kernel* (Eq 2.4).

$$k(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1 & \text{for } \lambda \geq |\boldsymbol{x} - \boldsymbol{y}| \\ 0 & \text{otherwie} \end{cases}, \tag{2.3}$$

$$k(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{(2\pi)^{d/2}} e^{\frac{1}{2}|\boldsymbol{x}-\boldsymbol{y}|}, \tag{2.4}$$

where d is the number of dimensions of $\boldsymbol{x}$

The density function for the whole dataset is then defined

$$f(\boldsymbol{x}) = \frac{1}{nh^d} \sum_{i=1}^{n} k\left(\frac{\boldsymbol{x}}{h}, \frac{\boldsymbol{x_i}}{h}\right), \tag{2.5}$$

where h is what is called the *bandwidth* that is used to adjust the "smoothness" of the density function, a higher bandwidth means fewer clusters but also potentially less clusters that occurs from noise in the data.

Using the density function; one can shift data points towards the peaks of this probability distribution. When all of the data points have converged on top of one of the peaks, clusters can be found by finding all overlapping data points using some small threshold radius.

2.4 Diffusion mapping

Diffusion maps are used for dimensionality reduction, or feature extraction. As opposed to PCA which is a linear dimension reduction technique, diffusion maps are non-linear and take into account multiple scales of the data to discover an accurate representation of the underlying data manifold from which the data was generated.

One can think of diffusion maps as modeling the spread of heat through some medium or taking a random walk through the data where nearby points are more likely to be visited. This can also be described using a Markov chain.

The most fundamental concept when using diffusion maps is the concept of connectivity. The connectivity of two data points is described using a probability- or kernel function. One of the most commonly used kernels is the Gaussian kernel

$$k(x, y) = e^{-\frac{\text{dist}^2(x,y)}{\sigma}}, \tag{2.6}$$

where the 'dist' function represents a distance metric between data points x and y.

To create a diffusion map, we first need to construct what is called the *diffusion matrix,* which is constructed from the kernel function by taking each pair of data points and applying the kernel to them;

$$L_{i,j} = k(x_i, x_j) \quad i, j \in [1..n], \tag{2.7}$$

where n is the total number of data points in the set.

When handling spatial graphs, such as potential rooms of a building, an extension to this diffusion matrix is to use an adjacency matrix $A_{i,j}$. The adjacency matrix represents a graph where a $A_{i,j} = 1$ means that there is an edge (i.e. an adjacency relationship) between nodes i and j. The diffusion matrix is then defined

$$L_{i,j} = \begin{cases} k(x_i, x_j) & \text{for } A_{i,j} = 1 \wedge i \neq j \\ 1 & \text{for } i = j \\ 0 & \text{otherwise} \end{cases}. \tag{2.8}$$

This is used to set non-adjacent nodes to zero (not connected) and self-loops to one (fully connected).

Now we introduce a normalization step on L with the α parameter. This step is used to control the effect of the data point density on the infinitesimal transition of the diffusion

$$L^{(\alpha)} = D^{-\alpha} L D^{-\alpha}. \tag{2.9}$$

The D matrix is a diagonal matrix that holds the sum of each row from L

$$D_{i,i} = \sum_{j=1}^{n} L_{i,j}. \tag{2.10}$$

We can now construct the Markov chain transition matrix

$$M = {D^{(\alpha)}}^{-1} L^{(\alpha)}, \tag{2.11}$$

where the new diagonal matrix is similarly defined

$${D^{(\alpha)}}_{i,i} = \sum_{j=1}^{n} L_{i,j}^{(\alpha)}. \tag{2.12}$$

This transition matrix can now be used to run the diffusion process by just taking larger and larger powers of M. The number of steps taken is denoted by t, which can be regarded as a time parameter.

The last step is to create an embedding from the transition matrix, where each data point gets embedded into Euclidean space. The embedding is created using the eigenvalues λ and eigenvectors ψ of the transition matrix M^t.

The embedding is defined as:

$$\Psi_{i,k} = \lambda_k \psi_{i,k} \quad i \in [1..n] \quad k \in [1..m] \quad \text{where } m = \min(n, m_{max}). \tag{2.13}$$

m_{max} is the maximum number of eigenvalues that are used, which acts as an upper bound on the number of dimensions in the diffusion space. The number of eigenvalues otherwise runs a risk of being very large.

3

Methods

This chapter gives a detailed overview of the proposed pipeline for automatic reconstruction of indoor spaces from 3D point clouds. Our method takes a 3D point cloud of a LiDAR scanned indoor environment as an input. The points belonging to the roof and floor are removed based on their normals, and irrelevant clusters of points are further removed using DBSCAN. Candidate walls are identified using RANSAC to find planar primitives. Bounded wall patches are then detected in the inliers of the planar primitives. These are projected onto the floor plane as 2D line segments, which are weighted based on the number of inliers. A cell graph is built from the weighted line segments, with the weights attributed to the edges. A diffusion map is used to identify how the cells are connected to form rooms, which are finally saved as polygons. The outline of the whole algorithm is shown in Algorithm 2 and a visual overview is seen in Figure 3.1. The following sections describe in more detail what is done in the different steps of the pipeline.

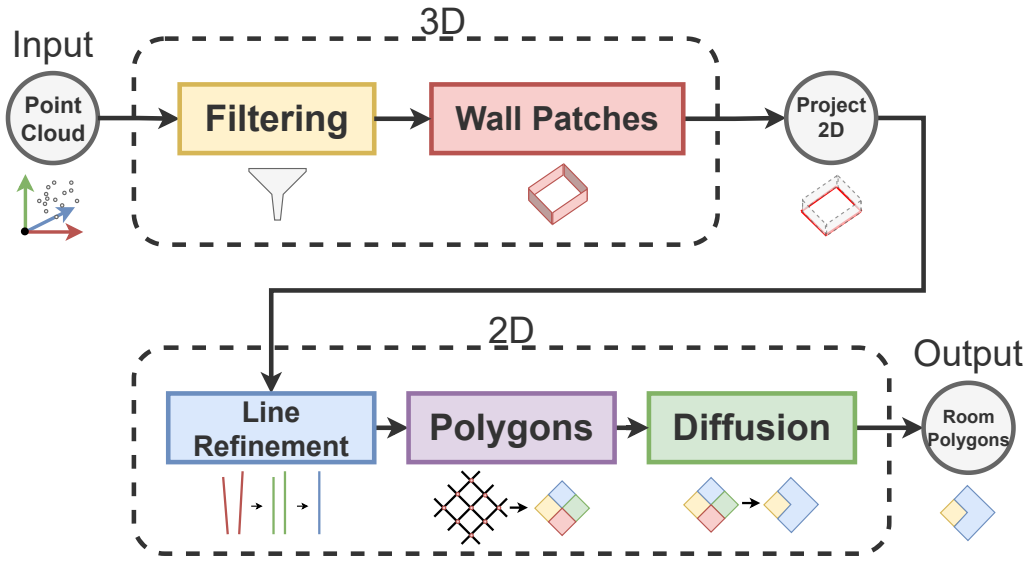


Figure 3.1: A visual overview of the major steps of the algorithm.

Algorithm 2: Pipeline subroutine overview

```
// Down sample and standardize point density
```
$\mathcal{D} \leftarrow \text{voxel\_downsample}(\mathcal{D}, \xi)$

```
// Filter noise, obstructions and other non-wall objects
```
$\mathcal{D} \leftarrow \text{filter\_outliers}(\mathcal{D}, \varepsilon_{\text{filter}})$

```
// Find plane primitives and wall patches (Section 3.1)
```
$\Pi_{\mathcal{W}} \leftarrow \text{ransac\_plane\_patches}(\mathcal{D}, \epsilon, \gamma, \varepsilon_{\text{patch}}, n_{\text{search}}, n_{\text{tries}})$

```
// Create line segments from the wall patches (Section 3.1.2)
```
$\mathcal{S} \leftarrow plane\_patches\_to\_segments(\Pi_{\mathcal{W}})$

```
// Cluster line segments by direction (Section 3.1.3)
```
$\mathcal{S} \leftarrow direction\_mean\_shift(\mathcal{S}, h_{\angle})$

```
// Cluster line segments by relative position (Section 3.1.3)
```
$\mathcal{S} \leftarrow position\_mean\_shift(\mathcal{S}, h_{\parallel})$

```
// Extend lines by adding a padding (Section 3.1.3)
```
$\mathcal{S} \leftarrow pad\_lines(\mathcal{S}, \eta)$

```
// Create a graph from the line segments (Section 3.2.1)
```
$G \leftarrow segments\_to\_graph(\mathcal{S})$

```
// Create polygons from the line segments (Section 3.2.2)
```
$P \leftarrow segments\_to\_polygons(\mathcal{S})$

```
// Calculate weights of the segment graph (Section 3.2.1)
```
$G \leftarrow weight\_calculation(G, n_{\text{bins/length}})$

```
// Create a cell graph (Section 3.2.2)
```
$C \leftarrow cell\_construction(G, P)$

```
// Calculate diffusion embedding (Section 3.2.2)
```
$\Psi \leftarrow diffusion(C, t, \sigma, m_{\max})$

```
// Cluster diffusion embedding into rooms (Section 3.2.2)
```
$P \leftarrow diffusion\_embedding\_to\_rooms(C, \Psi, \varepsilon_{\text{diffusion}})$

```
// Remove small polygons
```
$P \leftarrow filter\_polygons(P)$

3.1 Wall Detection and 2D Projection

In the first phase of the data pipeline, the general layout of the point cloud environment is defined. Planar primitives are detected using RANSAC. Inlier clusters to the planes are classified as wall patches if the height and area requirements are met. The wall patches are then projected onto the floor plane as 2-dimensional line segments.

3.1.1 Robust Wall Patch Detection

The RANSAC algorithm is often used to detect shapes in point clouds and is used in this work to detect planar primitives (henceforth referred to as "planes" for simplicity). The equation of a plane is solved using a sample of three points, and is then scored by counting the set of inliers to it, i.e. the points p from the point cloud D within a predefined threshold distance ϵ to the plane.

The points are partitioned using a k-d tree to efficiently find the nearest neighbors. This idea is first proposed by Schnabel et al. [10]. A point in the point cloud is sampled randomly, and RANSAC is run on the nearest neighbors of it. RANSAC will identify the plane with the highest amount of inliers, being the set of points in the neighborhood defined by the k-d tree, with distance less than ϵ. The number of RANSAC iterations run to identify the plane with most inliers is the parameter n_{iter}. The normal of the plane is then used to deem if it can represent a wall. If the vertical component of the normal is close to zero, the plane is a possible wall. The algorithmic explanation for this step if found in Algorithm 3.

Algorithm 3: $\Pi_{\mathcal{W}} \leftarrow$ ransac_plane_patches$(\mathcal{D}, \epsilon, \gamma, \varepsilon_{\text{patch}}, n_{\text{search}}, n_{\text{tries}})$

$\mathcal{O} \leftarrow \mathcal{D}$
$\Pi_{\mathcal{W}} \leftarrow \varnothing$
while $|\mathcal{O}| > n_{search} \wedge k_{tries} < n_{tries}$ **do**
  $p_R \in_R \mathcal{O}$ ▷ Select random point from point cloud
  $\mathcal{N}(\mathcal{O}, p_R, \gamma) \leftarrow \{p \mid p \in \mathcal{O} \wedge |p_R - p| < \gamma\}$ ▷ Calculate nearest neighbors
  $\pi \leftarrow \text{ransac}(\mathcal{N}(\mathcal{O}, p_R, \gamma), \epsilon, n_{iter})$ ▷ Find plane from nn-cloud
  if $\hat{n}(\pi) \cdot \hat{k} > 0.5$ **then** ▷ Check if plane is vertical using normal of the plane and the basis vector for the z-axis $\hat{k}$
    $k_{\text{tries}} \leftarrow k_{\text{tries}} + 1$
    continue
  end
  $\mathcal{P} \leftarrow \{p \mid p \in \mathcal{O} \wedge |(p - r_o(\pi)) \cdot (\hat{n}(\pi))| < \epsilon\}$ ▷ Calculate plane inliers for the whole cloud
  $k_{\text{patches}} \leftarrow 0$
  foreach $\mathcal{W} \in \text{dbscan}(\mathcal{P}, \varepsilon_{patch}, MinPts = 1)$ **do**
    $\mathcal{W}_{2D} \leftarrow \{(p \cdot v_1(\pi), p \cdot v_2(\pi)) \mid p \in \mathcal{W}\}$ ▷ Project the inlier points onto the plane giving 2D points
    $A \leftarrow \text{area}(\text{convex\_hull}(\mathcal{W}_{2D}))$ ▷ Get area of convex hull of the 2D points in the wall patch
    if $A > A_{min}$ **then**
      $\Pi_{\mathcal{W}} \leftarrow \boldsymbol{\mathcal{W}} \cup \{(\pi, \mathcal{W}_{2D})\}$ ▷ Plane and point cluster to set of wall patches
      $\mathcal{O} \leftarrow \mathcal{O} \setminus \mathcal{W}$ ▷ Remove wall patch from outlier set
      $k_{\text{patches}} \leftarrow k_{\text{patches}} + 1$
    end
  end
  if $k_{patches} = 0$ **then**
    $k_{\text{tries}} \leftarrow k_{\text{tries}} + 1$
  end
end

3.1.2 Wall Patch Segmentation and Refinement

The inliers to the vertical plane in the whole cloud is then calculated. Then they are clustered using DBSCAN to find bounded patches of inliers on the plane found by RANSAC. We calculate the area of the inlier clusters by projecting the points onto the plane and computing the convex hull of each cluster. Both the projected points and convex hull are in 2D. If the area is larger than A_{min}, the 2D point cluster is saved as a wall patch and removed from the search space. See Figure 3.2 for an illustration of the wall patches.

The search for wall patches ends when there are too few points left in the search space (less than n_{search}), or when no wall patch has been found for a set number of steps (n_{tries}).

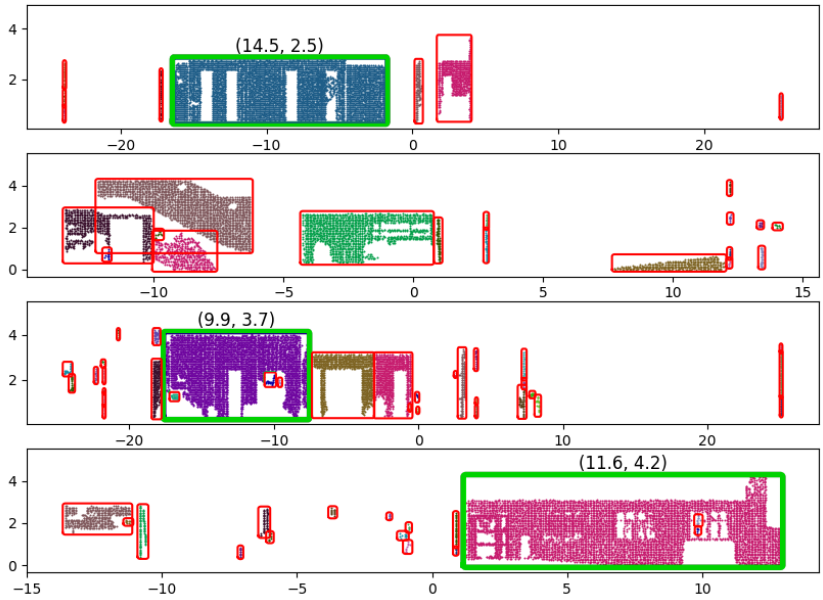


Figure 3.2: Illustrating 2D projection of selected point clusters, each cluster is assigned a random color. Clusters with area below A_{min} are marked with red and accepted wall patches are marked with green. The selected green patches is labeled with its size (width, height) in meters.

The wall patches are then projected onto the floor plane, yielding lines in 2D. Bounds of the lines are calculated using the wall patch points to give line segments $\mathcal{S}$, see Algorithm 4 for the complete calculations or Figure 3.3 for a visualization of this step.

Algorithm 4: $\mathcal{S} \leftarrow plane\_patches\_to\_segments(\Pi_{\mathcal{W}})$

$\mathcal{S} \leftarrow \varnothing$ ▷ Create empty line segment set
foreach $(\pi, \mathcal{W}_{2D}) \in \Pi_{\mathcal{W}}$ **do**
  $T(l) \leftarrow \hat{k} \times \hat{n}(\pi)$ ▷ Take cross product between basis vector for z-axis $\hat{k}$ and the normal for the plane to get the tangent vector for the line that intersects floor and wall plane
  $\boldsymbol{A} \leftarrow \begin{bmatrix} \hat{k} & \hat{n}(\pi) & T(l) \end{bmatrix}$ ▷ Create a matrix with the three direction vectors
  $\boldsymbol{d} \leftarrow \begin{bmatrix} 0 \\ \pi_d \\ 0 \end{bmatrix}$ ▷ Vector with d parameter from the general form of π
  $\boldsymbol{A}l = \boldsymbol{d} \Rightarrow l \leftarrow \boldsymbol{d}\boldsymbol{A}^{-1}$ ▷ Solve the system for the intersection line
  $\alpha \leftarrow \min\{p_x \mid p \in \mathcal{W}_{2D}\}$ ▷ Find the minimum bound of the line using its patch
  $\beta \leftarrow \max\{p_x \mid p \in \mathcal{W}_{2D}\}$▷ Find the maximum bound of the line its patch
  $a \leftarrow r_{0xy}(\pi) + \alpha T_{xy}(l)$ ▷ Calculate the start point of the segment using plane center and direction
  $b \leftarrow r_{0xy}(\pi) + \beta T_{xy}(l)$ ▷ Calculate the end point of the segment
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{(a, b)\}$ ▷ Add the segment to the set of segments
end

From now on we work with data in 2D only. Hence when discussing points in the point cloud we assume their 2D projections onto the floor plane.

3.1.3 Clustering line segments

The line segments are refined using mean shift. The mean shift refinement is done in two separate steps, first based on the direction of the lines, and then based on their position and distance to one another. See Figure 3.4 for an illustration that explains and compares the methods.

In the first refinement which is based on the direction of the lines, lines with similar direction are found and combined into clusters. All lines in each cluster are then rotated so that the direction becomes equal to the mean of the directions within the cluster. See Algorithm 5 for the calculations in this step, and Figure 3.4b for a visual explanation. The direction-based clustering ensures that lines with a similar direction become aligned to have the same direction. This removes inaccuracies in the line directions that may be caused by occlusions in the point cloud. When the lines are aligned, all lines within each clusters have the same direction.

The second refinement is performed based on the relative distance of parallel lines. Lines that are within the same direction based cluster, i.e. parallel, and close to

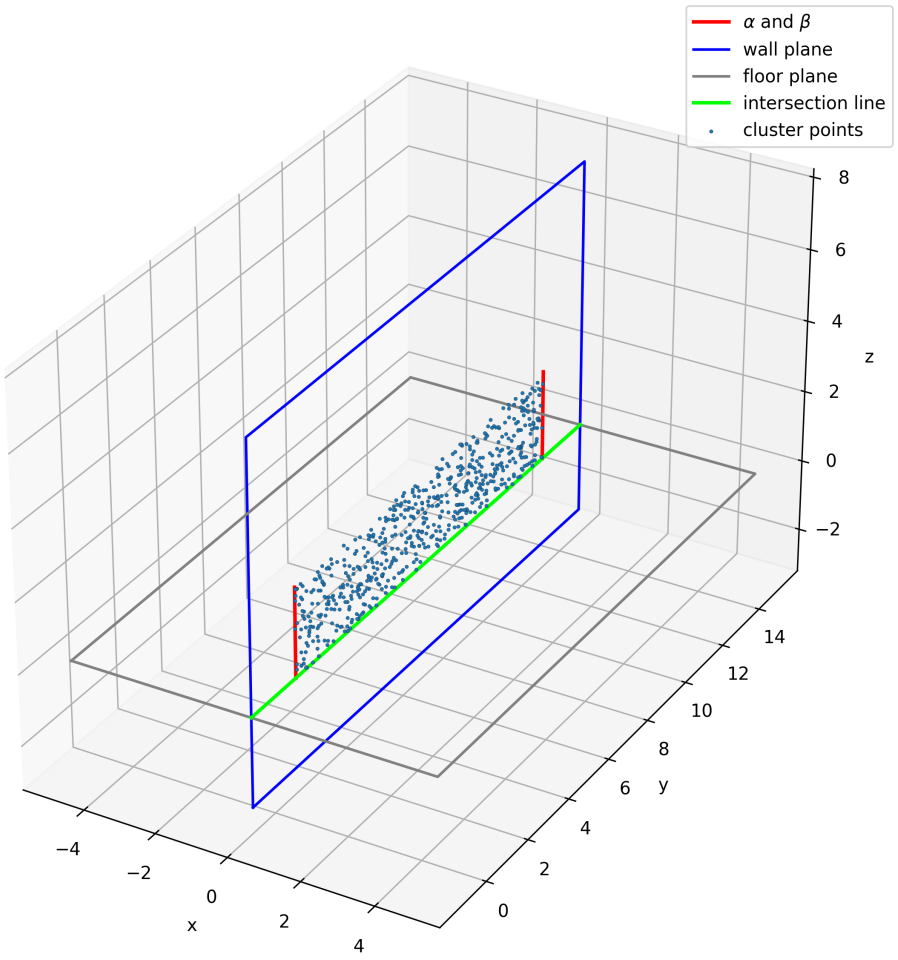


Figure 3.3: Illustrating the detected wall patch points in blue, its RANSAC wall plane in blue, the floor plane in gray, the intersection line between the RANSAC plane and floor plane in green and the minimum and maximum bounds (α and β) in red.

each other are merged. An average line position of the lines that are being merged is calculated and weighted by the number of inliers to the lines. All of the lines in the cluster are then translated to this position and overlapping lines are merged.

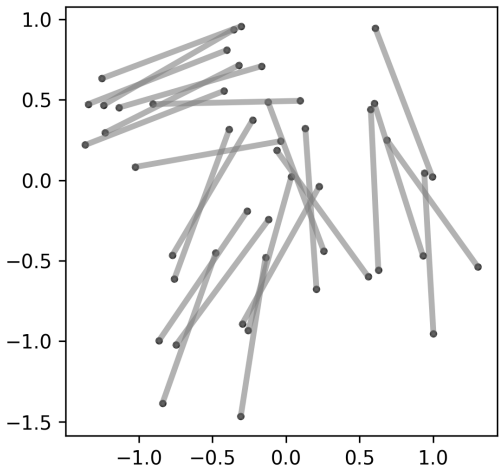
A padding η is finally added to the lines to compensate for missing data. The padding ensures that lines which correspond to real walls intersect where they should, so that polygonizaton of the line segments is possible. See Algorithm 6 for the calculations of this step, and Figure 3.4c for an illustration of it.

Algorithm 5: $\mathcal{S} \leftarrow direction\_mean\_shift(\mathcal{S}, h_\angle)$

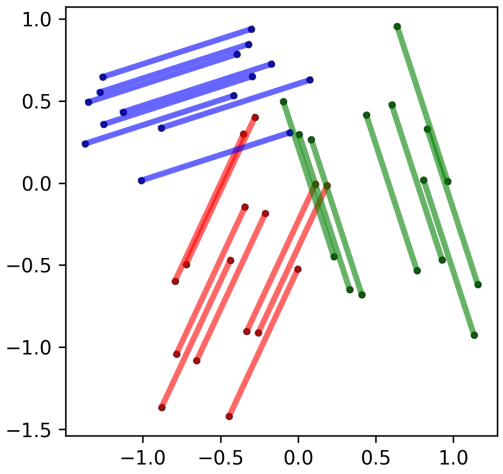
$\Delta_k \leftarrow b_k - a_k \quad$ where $(a_k, b_k) \in \mathcal{S}$ ▷ Calculate vector between start/end
$\theta_k \leftarrow \arctan2(\Delta_{ky}, \Delta_{kx})$ ▷ Calculate angle of segment vector
foreach $\mathcal{I} \in$ mean_shift$(\theta, h_\angle)$ **do** ▷ For each set of cluster indices
  $\theta_{\text{mean}} \leftarrow \sum_{i \in \mathcal{I}} \theta_i / |\mathcal{I}|$ ▷ Calculate the mean angle in cluster
  for $i \in \mathcal{I}$ **do** ▷ Set the mean angle for each angle in cluster
    $\theta_i \leftarrow \theta_{\text{mean}}$
  end
end
$(a_k, b_k) \leftarrow$ rotate_segment$(\mathrm{a_k}, \mathrm{b_k}, \theta_\mathrm{k})$ ▷ Rotate segment using shifted angle

Algorithm 6: $\mathcal{S} \leftarrow position\_mean\_shift(\mathcal{S}, h_\parallel)$

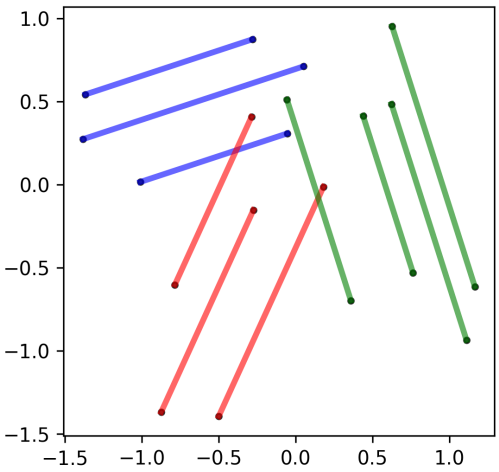
foreach $\mathcal{S}_\parallel \in$ group_parallel_segments$(\mathcal{S})$ **do**
  $\Delta_k \leftarrow b_k - a_k \quad$ where $(a_k, b_k) \in \mathcal{S}$
  $v_k \leftarrow (-\Delta_{ky}, \Delta_{kx})$ ▷ Calculate perpendicular vector
  $l_k \leftarrow |v_k|$ ▷ Calculate length of segment
  $\hat{n} \leftarrow v_0 / l$ ▷ Normalize perpendicular vector
  $\alpha_k = a_k \cdot \hat{n}_k$ ▷ Calculate the position of the line relative to normal
  $w_k = l_k / \sum_{i=1}^{n} l_i$ ▷ Calculate weights based on length of segments
  $\alpha_{\text{mid}} = \sum_{i=1}^{n} w_i \alpha_i / n$ ▷ Do weighted sum to get new line position
  $\delta_k = \alpha_{\text{mid}} - \alpha_k$ ▷ Calculate the shift for each segment
  $a_k \leftarrow a_k + \hat{n}\delta_k$ ▷ Shift the segment to the midpoint
  $b_k \leftarrow b_k + \hat{n}\delta_k$
end



(a) Input lines to the clustering steps



(b) After direction-based mean shift



(c) After position-based mean shift

Figure 3.4: Example of line segments before mean shift **(a)**, after being shifted by direction **(b)** and after being shifted and merged by position **(c)**. The colors corresponds to clusters created by the mean shift algorithm.

3.2 Room detection

A cell complex is built by creating polygons from the intersecting line segments. A cell complex is a set of non-intersecting polygons defined by a set of intersecting lines. Each cell in our complex represents a room or a partial room. The cell complex is used as a basis for a diffusion mapping. Cells are merged through diffusion and clustering to form the final room spaces. This section describes these steps in more detail.

3.2.1 Weighted graph construction

From the line segments a graph G is built with the line intersection points as nodes and the lines between them as edges (see Figure 3.5a to 3.5c). The nodes of the graph contain the coordinates of their location as attributes (x- and y-coordinates on the floor plane). Based on the number of inliers, a weight is assigned to each edge. The weights are calculated using a partitioning of edge sub-segments. If a sub-segment has inliers above a certain threshold, it is considered to be covered. The weight assigned to each edge is calculated as $w_{ij} = \frac{n_{cov}}{n_{tot}}$, with n_{cov} being the amount of covered sub-segments and n_{tot} being the total amount of sub-segments in the edge, which is 16 times the edge length, in meters. Concretely, a high weighted edge is likely to correspond to a real wall, since a large part of it is covered by points.

3.2.2 Cell complex graph diffusion

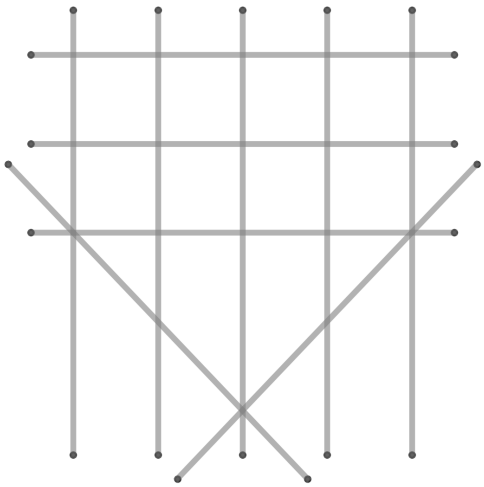
The weighted graph described in the previous subsection is then remade into a cell complex. Still represented as a graph, each node now corresponds to a cell, which is a polygon bounded by the surrounding line segments. Each cell represents a space; i.e. a possible room or a part of a room (see Figure 3.5c to 3.5d). The edges represent candidate walls and keep the coverage weights calculated in the last step. A diffusion map is created based on these weights, much as the one in the work by Mura et al [15].

An affinity matrix $\mathbf{L}$ is created using the coverage weights. Rather than measuring the similarity between cells, it represents how much of the edge between them is covered, creating an artificial distance in the diffusion space. The matrix entries are calculated as follows, with $i, j \in 1, \ldots, n$ where n is the number of cells, with c_i and c_j being cells at indices i and j, and w_{ij} being the coverage weight of the corresponding edge. See Algorithm 7 for the complete calculations.

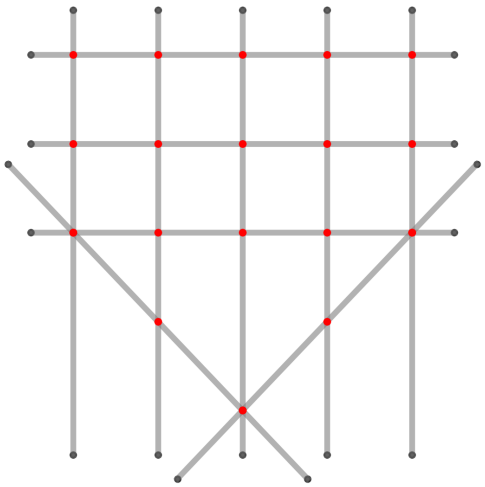
Algorithm 7: $\Psi \leftarrow diffusion(C, t, \sigma, m_{\max})$

$k(x, y) = e^{-\frac{w^2(x,y)}{\sigma}}$ ▷ Gaussian kernel with weight between cells as distance function
for $i \in [1..n]$ **do** ▷ Iterate i for n number of cells
  for $j \in [1..n]$ **do** ▷ Iterate j for n number of cells
    $L_{i,j} = k(C_i, C_j)$ ▷ Initialize the L matrix using kernel and cell graph
  end
  $L_{i,i} = 1$ ▷ Set diagonal elements to one
  $D_{ii} = \sum_{k=1}^{n} L_{ik}$ ▷ Initialize diagonal matrix
end
$M = D^{-1}L$ ▷ Create Markov chain transition matrix
$\lambda, \psi \leftarrow \text{eig}\,(M^t)$ ▷ Calculate eigenvalues and vectors at timestep t
$m \leftarrow \min\{n, m_{\max}\}$ ▷ Limit the number of eigenvalues to maximum $m_{\max}$
for $i \in [1..n],\ k \in [1..m]$ **do**
  $\Psi_{i,k} = \lambda_k \psi_{i,k}$ ▷ Calculate the diffusion embedding
end

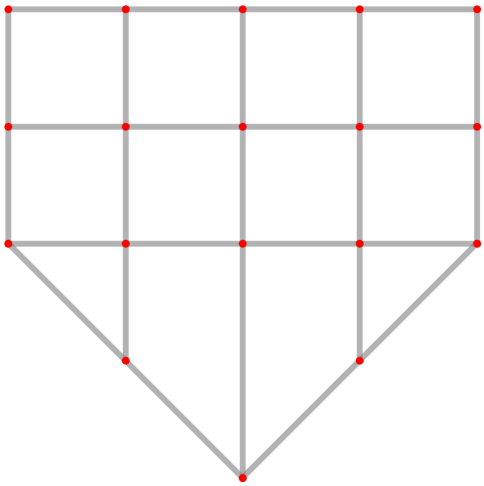
The **L** matrix is then used to create a Markov transition matrix **M**, representing the chance or degree to which a diffusing element would spread through the space defined by the cell matrix. Candidate rooms are formed based on which cells have a high level of adjacency in the diffusion space, where adjacent cells with low weighted edges are likely to be fused.



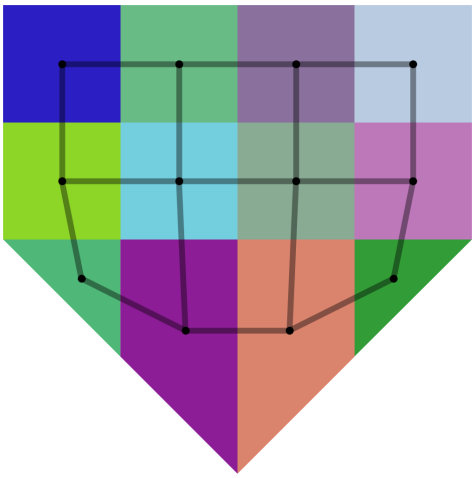
(a) Input line segments



(b) Finding line intersections



(c) Constructing segment graph



(d) Constructing cell complex graph

Figure 3.5: Example line segments showing the process to create both the segment graph and the cell complex. Starting with line the line segments in **(a)**, intersection points are found **(b)** shown in red, the intersection points are used to construct a segment graph used to calculate a weighted graph **(c)**, the final step creates a cell complex **(d)** where the cells becomes the nodes.

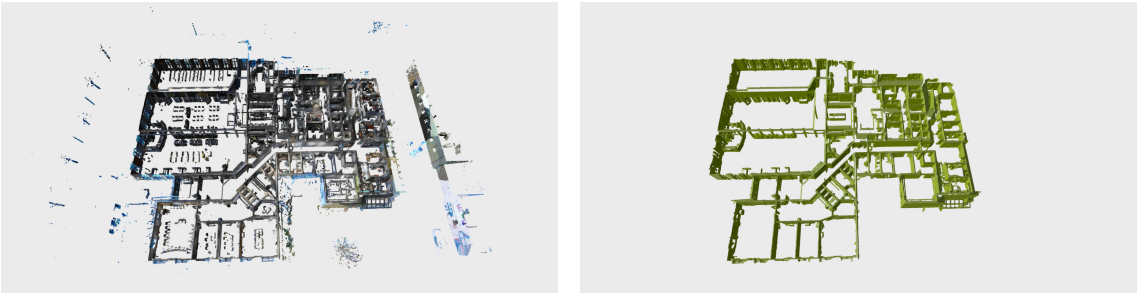
3.2.3 Cell Merging via Clustering

The cells are then clustered using the embedding $\Psi_{i,j}$ from the diffusion map. Cells that are in the same room will be closer in the diffusion embedding space, which means that it is possible to extract the rooms using a clustering algorithm such as DBSCAN. The cells that are in the same cluster are then merged to create a single polygon for the whole room.

3.3 Pre- and post-processing

A random sample of the point cloud was taken since the whole cloud would be too large to handle in a computer with regular memory capacity. The point cloud is then center shifted, i.e. a global translation is applied to the cloud so that the center of mass of the point cloud is at the origin of the coordinate system. Downsampling of the point cloud is done through a voxel down sampling step, to standardize the point density and yield a manageable amount of points without losing too much accuracy. Two statistical outlier removal steps are also run, to get rid of most of the noise efficiently. Only the walls of the buildings were deemed necessary for space classification. Therefore, both roof and floor points are identified by their approximate normals and removed as outliers. This classification and removal step is done between the statistical outlier removal steps.

The point cloud is further filtered by clustering. Clusters are created with DBSCAN, using the spatial point density, and bounding boxes are then created for each cluster. Any cluster that is shorter than a height threshold based on the height of the walls is then removed. The separate clusters that remain are then merged together again into one point cloud. Illustrations comparing the point cloud before and after the filtering steps are seen in Figure 3.6.



(a) Point cloud before filtering

(b) Point cloud after filtering

Figure 3.6: One of the point clouds from the dataset at Zynka, with roof and floor points removed, is shown before and after the filtering steps. Lots of clutter and objects both inside and outside the building can be seen, which is removed through the filtering.

The final step is to convert the room polygons into WGS84 coordinates and into GeoJSON format that can be visualized on a map. This is done using a library

called PyPROJ that specializes in converting between different coordinate systems.

The point clouds are given in a local coordinate system based around a project origin, which is the same geographical point for all buildings in a project. The point clouds are offset from the local Oreg, which is a translation that has to be reversed for the final output to have accurate real-world coordinates. This offset is read from the point cloud file and applied at the end of the pipeline.

3.4 Implementation Details and Input Data

The source code was written as a modular data pipeline in Python. Libraries used include Open3D for the point cloud handling, NetworkX for the graphs, and Shapely for the polygons.

The point clouds used were given in the e57 format, which was converted to PLY for use with the Open3D library. Ten point clouds were initially supplied by Zynka for the project, of which 3 had corresponding manually created polygon models. Each point cloud consisted of around 100 million points and represented a single floor of a building, with an area ranging from 500 to 2000 square meters. Random samples of 10 million points were taken from the clouds to fit in the memory of the computer we used (see Table 3.1) and improve runtime performance significantly.

All of the parameter values used for the implementation are shown in Table 3.2. A thorough search for optimal parameter values has not been performed due to time constraints. The parameters of inlier distance ϵ and the DBSCAN parameters were set to values that were deemed reasonable based on the thickness of walls inside of buildings. The parameters set for voxel downsampling depended to a large degree of the runtime performance. The parameters for mean shift and diffusion were decided on through experimentation to maximize performance and robustness.

Table 3.1: The specifications of the computer used for running the algorithm.

Spec	Value
CPU	Intel(R) Core(TM) i7-3770K CPU @ 3.50GHz 4 Cores
RAM	16GiB - 4x4GiB DIMM DDR3 Synchronous 1333 MHz

Table 3.2: All of the parameter values used when generating the results.

Parameter	Value	Description
ξ	0.08	Voxel down sampling voxel size
ϵ	0.05	Inlier distance threshold
γ	4.0	Point cloud sample nearest neighbor radius
$\varepsilon_{\text{filter}}$	0.12	DBSCAN maximum neighbor distance for filtering
$\varepsilon_{\text{patch}}$	0.15	DBSCAN maximum neighbor distance for wall patches
$\varepsilon_{\text{diffusion}}$	0.01	DBSCAN maximum neighbor distance for room clustering
$MinPts$	5	DBSCAN minimum cluster points
η	2.0	Line segment padding
$h_{\angle}$	0.2	Mean shift direction bandwidth
$h_{\|\|}$	0.2	Mean shift position bandwidth
t	13	Diffusion time steps
σ	0.1	Diffusion weight scaling factor
$m_{\max}$	80	Number of eigenvectors used for diffusion embedding

4

Evaluation and Results

This section presents the results and evaluation of the generated models.

4.1 Evaluating against human labeled polygons

Our dataset includes human labeled 2D polygonal models that have been created using the same input point clouds as the automatically generated models. A plot comparing two such models is shown in Figure 4.2.

Set-theoretic operations are applied to the two models to evaluate the quality of the generated polygons compared to the human labeled polygons. The first operation is set-difference, where we can see added and missing area between the two models. A visualization of the set-difference can be seen in Figure 4.3.

The second evaluation metric is Intersection over Union (IoU) where the area of the intersection between the polygon sets is calculated and divided by the area of the union between them. An example intersection and union are shown in Figure 4.4.

4.1.1 Large building

Figure 4.1 shows a relatively large building that the prototype is evaluated on. It was chosen due to the large size and high level of complexity, noise and occlusions.

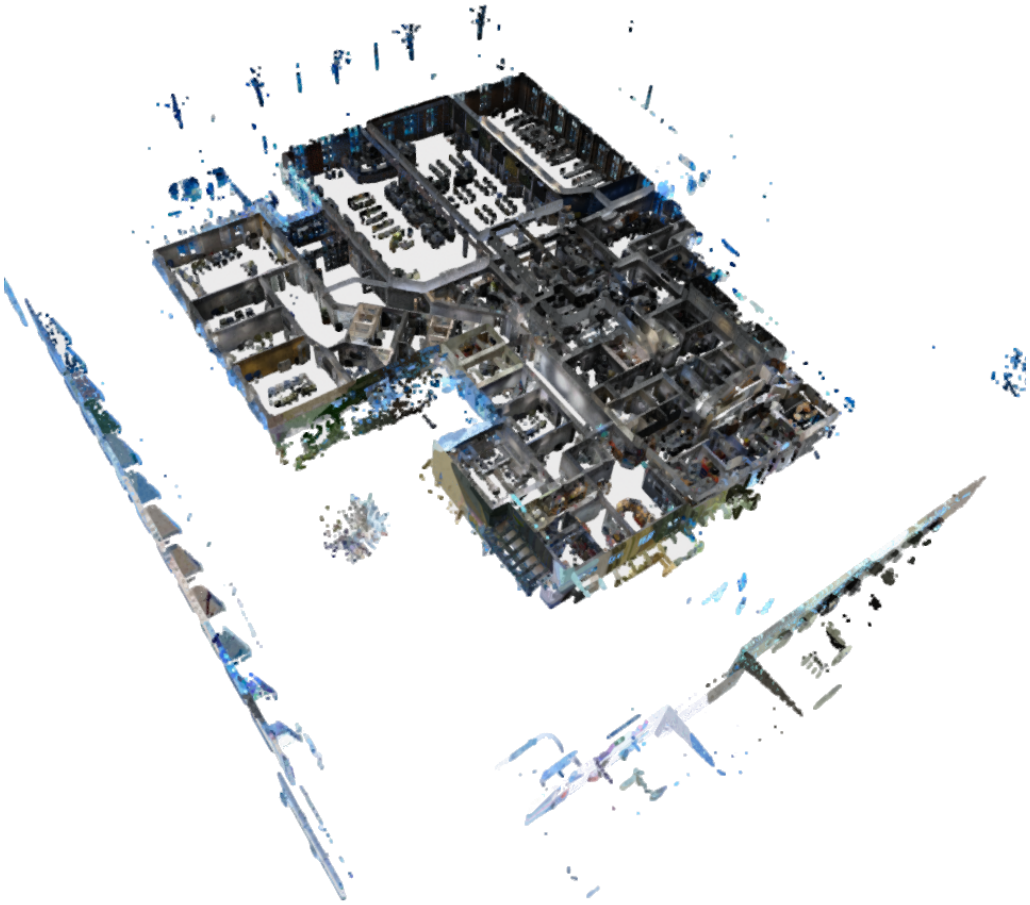


Figure 4.1: The larger building point cloud, showing a lot of clutter, reflections and obstructions.

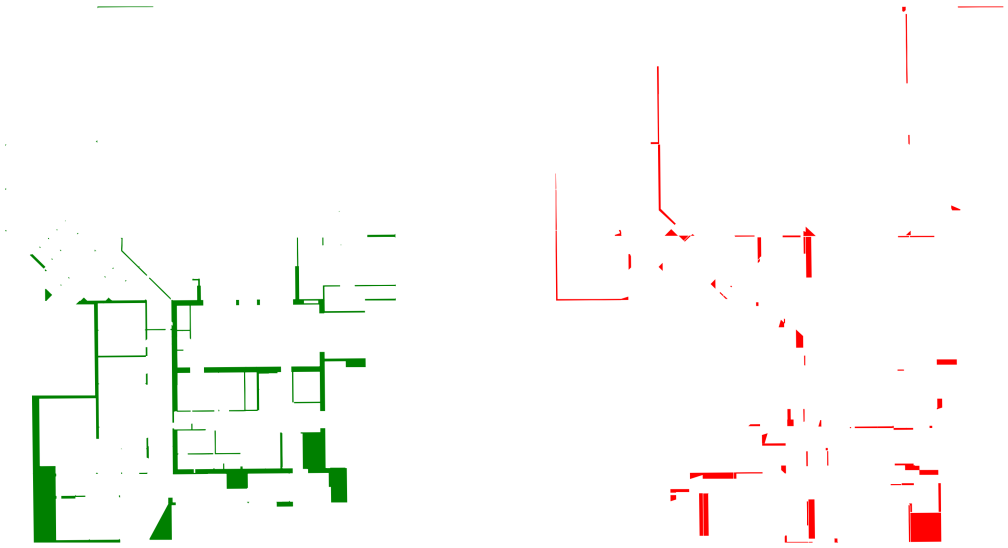
In Figure 4.2 is a comparison of our generated polygons to the and the human labeled polygons. We can see that most rooms are captured by our algorithm but some extra rooms are created. We get around 20 extra walls on average for this building (see Table 4.1).

(a) $P_{\text{generated}}$ **(b)** P_{human}

Figure 4.2: Comparing the generated room polygons $P_{\text{generated}}$ **(a)** with the human labeled P_{human} **(b)** polygons.

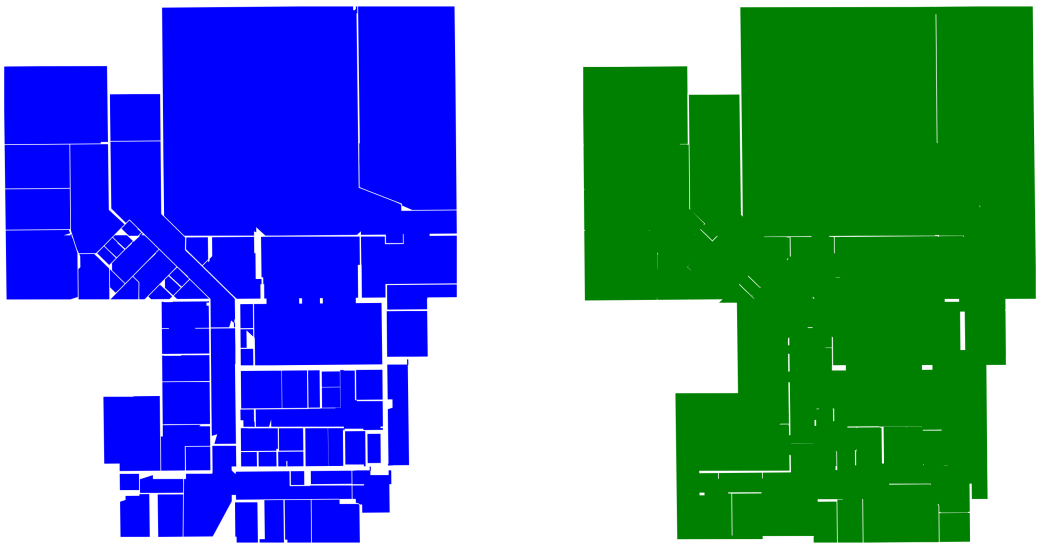
The set theoretic difference between the generated and human labeled polygons is visualized in Figure 4.3. The additional area created by our algorithm is seen in 4.7a. In Figure 4.7b the missing area from our algorithm's output compared to the human labels is shown. Table 4.1 shows that around $76m^2$ of additional area is generated and $46m^2$ is missing on average, which means that our algorithm tends to create more additional area rather than missing area for this particular building.

The Intersection Over Union metric is commonly used to measure the extent of overlap between two different shapes. It is calculated by taking the area of intersection between the two sets of polygons and dividing by the area of the union. The maximum score is 1.0 and this means that the shapes are identical. See Figure 4.4 for a visualization of the intersection and union between the generated and human labeled polygons. The Root Mean Squared distance from each polygon to the points in the point cloud was also measured, which shows the approximate average error in terms of distance to the points.



(a) $P_{\text{added}} = P_{\text{generated}} \setminus P_{\text{human}}$ **(b)** $P_{\text{missing}} = P_{\text{human}} \setminus P_{\text{generated}}$

Figure 4.3: Comparison between the additional area P_{added} **(a)** and the area that is missing P_{missing} **(b)** relative to the human labeled rooms. In the additional area plot one can see rooms and dividing walls that were created but should not exist. One missing room that was not found by the algorithm can be seen in the missing area plot.



(a) $P_{\text{intersection}} = P_{\text{generated}} \cap P_{\text{human}}$ **(b)** $P_{\text{union}} = P_{\text{generated}} \cup P_{\text{human}}$

Figure 4.4: The intersection **(a)** and the union **(b)** of the generated and human polygons.

Table 4.1: The evaluation metrics over 6 runs of our algorithm on the large building. The arrows indicate whether a higher or lower value is favourable.

Name	Mean	STD
Missing Area↓	46.19 m^2	±15.574
Additional Area↓	76.027 m^2	±22.02
Additional Rooms↓	20.333	±4.46
Intersection Area↓	1414.158 m^2	±15.399
Union Area↓	1605.785 m^2	±22.268
Intersection Over Union↑	0.881	±0.015
RMS distance to points↓	0.051 m	±0.002
Number of Runs		6

4.1.2 Small building

Figure 4.5 shows a small building that is also used for evaluating the performance of the prototype.

We evaluate this building using the same methods as with the previous building. See Figure 4.6 for the polygon comparison, Figure 4.7 for the added and missing area, Figure 4.8 for the intersection and union and finally Table 4.2 for the numerical results.

(a) $P_{\text{generated}}$ **(b)** P_{human}

Figure 4.6: Comparing the generated room polygons $P_{\text{generated}}$ **(a)** with the human labeled P_{human} **(b)** polygons.

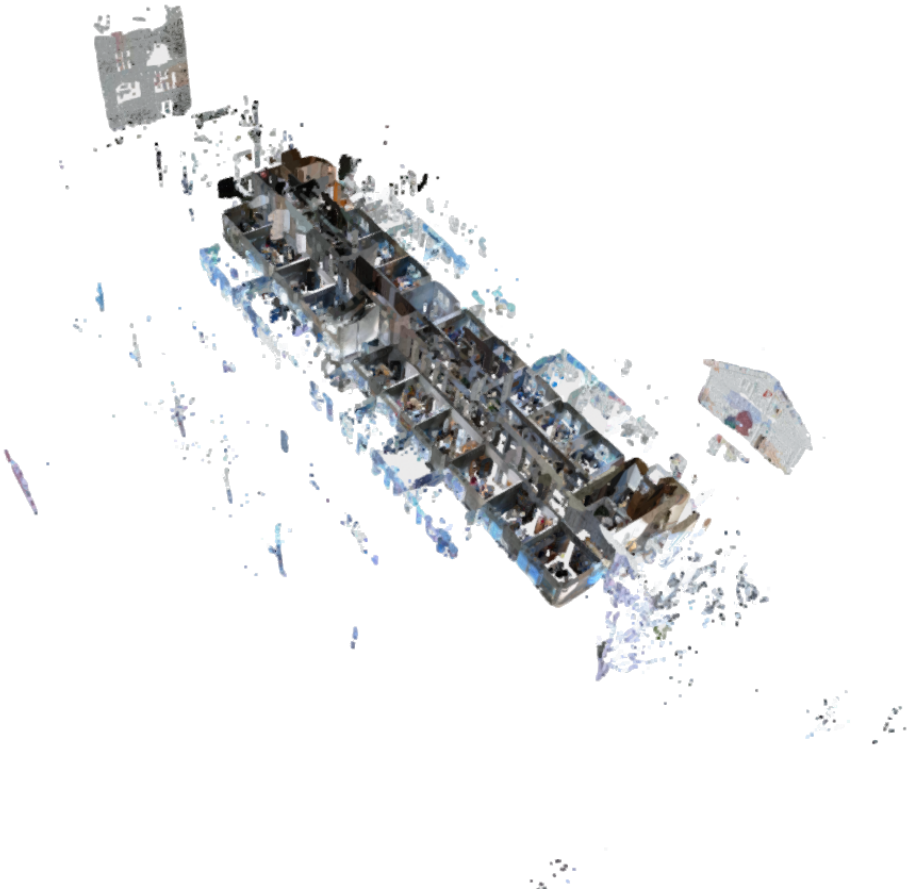
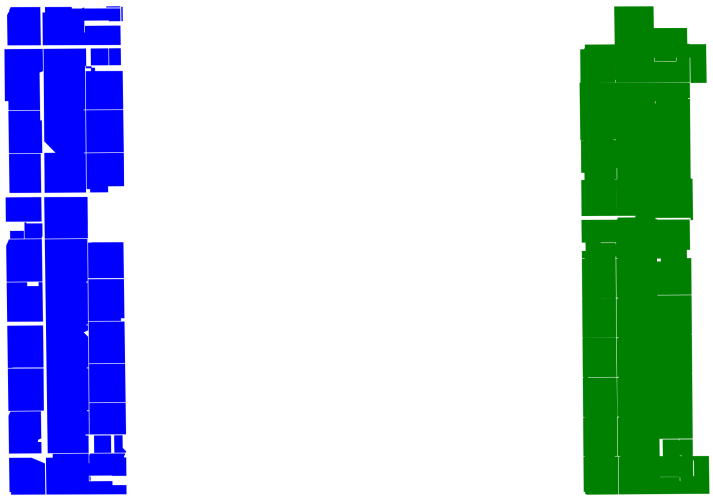


Figure 4.5: The smaller building point cloud, also with a lot of clutter, reflections, obstructions and even other buildings outside the windows.

(a) $P_{\text{added}} = P_{\text{generated}} \setminus P_{\text{human}}$ **(b)** $P_{\text{missing}} = P_{\text{human}} \setminus P_{\text{generated}}$

Figure 4.7: Comparison between the additional area P_{added} **(a)** and the area that is missing P_{missing} **(b)** relative to the human labeled rooms. In the additional area plot one can see rooms and dividing walls that were created but should not exist. A few additional rooms that does not exist in the human labeled model were found by the algorithm and can be seen in the additional area plot.



(a) $P_{\text{intersection}} = P_{\text{generated}} \cap P_{\text{human}}$ **(b)** $P_{\text{union}} = P_{\text{generated}} \cup P_{\text{human}}$

Figure 4.8: The intersection **(a)** and the union **(b)** of the generated and human polygons.

Table 4.2: The evaluation metrics over 6 runs of our algorithm on the smaller building. The arrows indicate whether a higher or lower value is favourable.

Name	Mean	STD
Missing Area↓	37.957 m^2	±7.498
Additional Area↓	55.818 m^2	±4.119
Additional Rooms↓	17.667	±2.134
Intersection Area↓	457.969 m^2	±8.202
Union Area↓	579.005 m^2	±5.484
Intersection Over Union↑	0.791	±0.016
RMS distance to points↓	0.049 m	±0.001
Run time↓	558.866 s	±64.352
Number of Runs		6

4.2 Evaluation using synthetic point clouds

A problem with comparing the generated polygons with the human labeled ones is that no real ground truth exists. A solution to this is using the human labeled polygons as ground truth by generating a synthetic point cloud from them, that can then be reconstructed with our method to polygons and compared to the true polygons.

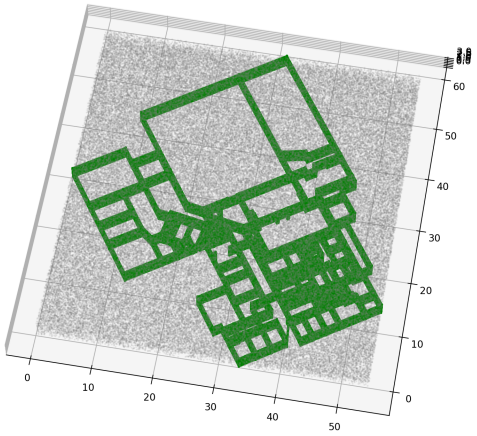
We create the point cloud by adding 3m high walls of random points for each edge of the polygons. Gaussian noise is added to the walls in an attempt to reproduce scanning inaccuracies. Random points are then added to the point cloud, corresponding to 5% of the points. See Figure 4.9a.

Reformulating the problem into a clustering problem, where the walls of a single room represents a cluster (see Figure 4.9b) gives the ability to evaluate the algorithm in these terms and use well established statistical metrics. This evaluation will focus on the performance of the RANSAC and wall patch step in the algorithm, since this is the most critical part to capturing the walls for each room.

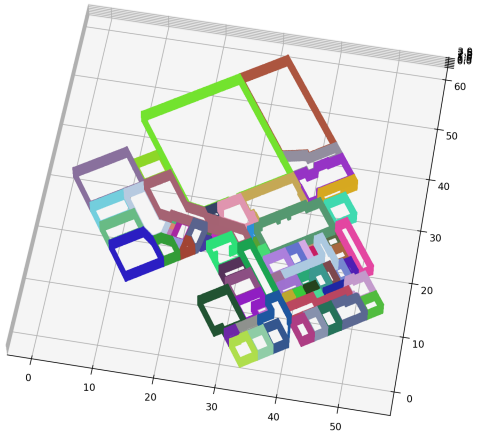
Four metrics were deemed suitable for this task. The recall and precision metrics evaluate the labeling performance, and the completeness and homogeneity scores evaluate the quality of the clusters. Recall is the most important score and can be intuitively explained as the ability of the algorithm to find all of the walls that are relevant for each room (the positive samples). See Figure 4.9d for a visualization of the failure cases. The precision score represents the ability of the algorithm to not create walls where there should not be any, i.e. not labeling clutter and doors as walls. The completeness score measures the ability of the algorithm to assign the ground truth room points to the same room cluster, while homogeneity measures the algorithms ability to create room clusters that do not overlap with other rooms in the ground truth, i.e. to what extent the room clusters are homogeneous. The measured values for the four metrics are found in Table 4.3.

Table 4.3: Metrics used for evaluating the performance of the algorithm relative to the ground truth.

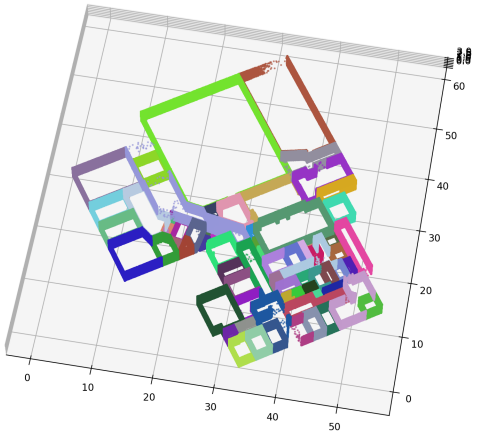
Name	Mean	STD
Completeness Score	0.86	0.011
Homogeneity Score	0.793	0.017
Precision	0.847	0.023
Recall	0.76	0.025
Number of Runs		3



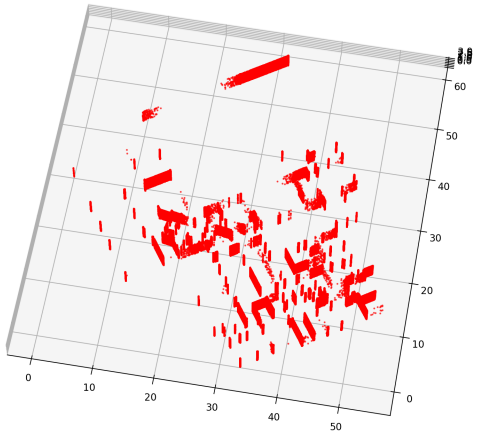
(a) Synthetic input



(b) Ground truth clusters



(c) Predicted clusters



(d) False negatives

Figure 4.9: The synthetic input point cloud **(a)** where the green points are walls and the gray points are 5% added noise, ground truth clusters **(b)** where each color represents a different room (or cluster), clusters generated by the algorithm **(c)** and the unlabeled walls (false negatives) **(d)** compared to the ground truth.

4.3 Runtime performance

The last metric used is the runtime of the algorithm, which is evaluated for the large building to around 22 minutes(see Table 4.4). The table also shows the runtime of all the different steps of the algorithm. The RANSAC step shows the highest standard deviation, which is likely caused by the random sampling of points. The rate at which the wall patches are found varies greatly because of the random plane sampling, especially since the planes are searched for in decreasing order of size.

Table 4.4: The runtime metrics of all steps over 4 runs of our algorithm on the large building. The steps that takes the longest is the reading of the cloud into memory and the ransac step.

Name	Mean (seconds)	STD (seconds)
Load point cloud	374.972	30.608
Voxel down sampling	2.899	1.975
Statistical outlier filter	2.162	0.331
Remove floor and ceiling	0.541	0.039
DBSCAN filtering	23.55	11.398
RANSAC wall candidate detenction	842.464	42.52
Directional Mean Shift on lines	1.437	0.607
Positional Mean Shift on lines	11.842	1.998
Line padding	0.792	0.027
Segments to segment graph	4.312	0.053
Find segment inliers	33.006	0.663
Calculate coverage weights	3.321	0.131
Construct Cell Complex Graph	4.956	0.293
Remaining	15.239	0.858
Total	1363.777	43.865
Number of Runs		4

5

Discussion

In this chapter the results of the project are discussed, and the additions to the field of research are described. Future improvements to the prototype are proposed, and the research methodology used for the project is commented on.

5.1 Comparison of Results and Expectations

This project is focused on reconstruction of the floor plan and dividing a building into individual rooms in the form of a 2D polygon model. The results of the project correspond well with the minimum viable product that was described prior to the start of the project, although a lot of improvements still need to be made to attain full use of the product. The corner points of rooms are however found to a high extent, and polygons are created from them to form a model that looks very similar to the ones created by hand by the employees at Zynka.

5.2 Discussion of Results

In the following subsections, the results and their implications are discussed.

5.2.1 Comparison with manual models

Pillars and other non-usable parts of the scanned indoor environments tend to be excluded from room boundaries in the manual model creation process at Zynka. Detecting these and excluding them from the output is necessary to gain full conformity between the models. The manually created spaces with the results of the prototype sometimes contain overlapping polygons. This gives rise to some issues when evaluating based on polygon comparison, as overlapping rooms in one model does not make sense, and are not accepted by the library (Shapely) used to compare the polygons.

5.2.2 Comparison with synthetic models

All of the scores from the clustering measurements in Table 4.3 go from 0 to 1 where 1 is the highest score. We can see that all of them are above 0.75, the precision is

higher than the recall which is to be expected, since RANSAC is known for being good at dealing with noise. Even though the recall is sufficiently high to let the algorithm create room polygons, it would be desirable to get it at least above 90% in the future to get an even more accurate model that captures all of the walls in a building. The completeness is higher than the homogeneity, our hypothesis on this result is that the algorithm sometimes creates extra rooms within a ground truth room, lowering the homogeneity of the room compared to the ground truth.

5.3 Additions to Research in the Field

Our work adds to the research in the field of indoor environment reconstruction by proposing a new combination of methods that yields a robust and accurate two-dimensional model of a whole building floor at a time. The algorithm runs on whole floors rather than single rooms, and can handle complex layouts with a number of diagonal walls. The prototype scales well with large buildings, especially since multiple walls in one plane can easily be found with RANSAC.

One key contribution of our work which we do not believe has been done before is the division of RANSAC plane inliers into clusters. Rather than keeping the whole planes and projecting them into 2D as infinite lines, which is done in the work by Mura et al. [15], we project the lines bounded by the wall segment bounds in the planes. Mura et al. use an arrangement of lines to create a cell complex of rooms, but by using infinite lines, far more cells are created than there are rooms due to all the extra intersections. This makes the processing of the cell complex slower than it would need to be. A problem with using bounded lines is however that they may fail to intersect due to partial loss of data, caused by occlusions for example. Our approach poses a compromise of sorts. We use bounded lines, but extend them by a padding to compensate for any eventual missing parts of the wall segments. The extension of the lines still yield more line intersections and cells than without extensions, but far fewer than when using unbounded lines. This makes processing of large building floors possible, even when there are lots of unusual wall angles. The prototype furthermore scales well with increasing size of buildings, since several walls which are on the same planar primitive are found simultaneously with RANSAC.

5.4 Experimental Uncertainties

One might argue that diffusion of the rooms should not be necessary when using bounded lines. However, with any extension of lines to cover gaps diffusion may still be a necessary step. There is still a bit of experimental uncertainty here, whether weighting and thresholding of the lines might be enough to prune away unnecessary ones, or if it can be said for certain that using diffusion to prune wall candidates is more useful or more robust. A comparison of the polygons before and after diffusion are shown in Figure 5.1 where we can see a large improvement of the room polygons.

(a) $P_{\neg\text{diffusion}}$ **(b)** $P_{\text{diffusion}}$

Figure 5.1: Comparing the generated room polygons before diffusion $P_{\neg\text{diffusion}}$ **(a)** with the polygons after running diffusion $P_{\text{diffusion}}$ **(b)**. Rooms that were divided by the extended wall patches is merged into single rooms, an example of this is the corridors in the middle-left of the building.

There is a trade-off between potentially losing candidate walls that would have existed if the lines had been unbounded, and avoiding the explosive growth in number of wall candidates that comes with using unbounded lines for buildings that have diagonal walls. Any diagonal wall that is represented with an unbounded line will intersect with a large number of horizontal and vertical wall representing lines. This yields a high amount of cell polygons, and made the processing extremely slow as compared to when bounding the lines. (10000's of rooms instead of 100's). Since a large amount of the scanned buildings in our point cloud dataset contain at least a few diagonal walls, we opted for using bounded lines to keep down the processing time. This however means that the prototype is a bit more sensitive to obstructions, as a largely occluded wall in the worst case is not detected at all. In the current version of the prototype, about 2% of rooms in the manually built model are missing in our output. We have several ideas on how this could be solved.

5.5 Future research and improvements

A main issue is that more planar surfaces than only walls, such as doors, come through to the final steps, and are weighted high enough to be counted as room dividing in the diffusion step. Some rooms are therefore split into several smaller rooms, which will have to be merged either manually or by more exact parameter adjustments. In other words, a common case is that a room has been divided into two parts or more, while the outer contours of the space remains true to the room

as drawn in the plan.

Future development of this product will involve scaling it to create 3D models of entire buildings, and not just the room models of one floor of a building at a time. As the prototype currently takes a split point cloud of only one floor, segmenting the building into floors and running the pipeline on each one will need to be added. Detection of staircases could be implemented as well, as well as possible recognition and labeling of larger, stationary furniture and utensils. Handling of thicker walls and removing the wall polygons will also need to be implemented. Thick walls risk being labeled as rooms, when they should really be removed from the model. To be able to extend the model to 3D in the future, we however need to save the walls and label them separately.

When using a mobile LiDAR scanner, a trajectory in form of a line is typically recorded along with the resulting point cloud. This trajectory, though not matching with any specific locations in the point cloud, can still be used to check what areas of the building were traversed, and what parts may be inaccessible. This could pose an option for dividing thick inner walls and other inaccessible areas from the rooms. For example, all spaces which the trajectory does not pass through may be labeled as wall or non-usable area.

An issue with the prototype is caused by large occlusions of walls in the point cloud, leading to the wall segments appearing to be far smaller than they really are, or missing completely. In these cases the line segments that correspond to the wall patches may not intersect with other walls where they should, despite the padding of the lines. No closed polygon can then be created to represent the room when the lines do not intersect, and hence the affected room spaces are not represented correctly, if at all, in the final model. Solving this may be a matter of patching together lines with dead ends which are close to intersecting with another line. Finding spots with possible wall occlusions and extending the line segments around those areas is another option. An even better option may be to run an extra step of RANSAC, with a broader scope. The planes found in this step may be pruned and only accepted in cases where no better wall candidate already exists.

A related idea for more efficient processing of the point clouds is to run RANSAC in the 2-dimensional plane, on the projected points. This should bring down the processing time significantly, possibly without loss of quality since all points and detected planes are projected to 2D directly after being labeled anyway. For optimization of the fit of the wall patches to the point cloud, the wall patches should be fitted to the points through for example linear least squares optimization. This fitting could be done in 2D with the line segments and projected points, which should make the step efficient. To create a full 3D model the walls however need to be found in 3D space, and other options for making the plane detection more efficient will need to be explored.

5.5.1 Evaluation methods

A performance comparison of our method to those developed in other recent works would be valuable. We contacted a few authors of projects that relate to ours but were not able to receive any source code for comparisons. Time constraints is the main reason that more authors were not contacted in the end. This kind of evaluation could however be done in the future.

5.6 Research methodology

The research and implementation methodology used in the project roughly followed the *Double Diamond* model [21]. Initially extensive research was done within the field to explore the existing methods and algorithms for point cloud manipulation such as outlier removal, plane detection, clustering and more. A suitable combination of approaches from the research was then chosen to begin the experimentation phase. The methods were implemented and tested. The prototyping phase was then commenced in which a prototype was built based on the results of the experimentation approach. Finally, the end to end product will be implemented, which is a work in progress. These steps corresponds to the *discovery*, *definition*, *development* and *delivery* steps that are defined by the double diamond model.

6

Conclusion

The goal of the project was to automate the steps of constructing 2D room-level polygonal models of a building floor from 3D point clouds. The prototype built in this work is a preliminary product, which is not yet fully reliable without human input. It generally detects all the rooms, but with a few more or less room divisions than in the corresponding manually built model. Solving this entirely is difficult since decisions such as what comprises a room, and what kind of walls are required to divide a space into two rooms is often quite subjective in nature. Because of this, human verification may be a necessary step even with a highly accurate automated product, especially if the model is to contain non-physical information such as room divisions that have been agreed upon with no physical wall in place. More data examples with manual annotations from different people are furthermore required for better evaluation of the proposed method.

Bibliography

[1] ISO, *Building Information Models: Information Delivery Manual. Methodology and Format*. International Organization for Standardization, 2016.

[2] R. Volk, J. Stengel, and F. Schultmann, "Building information modeling (bim) for existing buildings—literature review and future needs," *Automation in construction*, vol. 38, pp. 109–127, 2014.

[3] D. Huber, B. Akinci, A. A. Oliver, E. Anil, B. E. Okorn, and X. Xiong, "Methods for automatically modeling and representing as-built building information models," in *Proceedings of the NSF CMMI Research Innovation Conference*, vol. 856558, NSF, 2011.

[4] P. Musialski, P. Wonka, D. G. Aliaga, M. Wimmer, L. Van Gool, and W. Purgathofer, "A survey of urban reconstruction," in *Computer graphics forum*, vol. 32, pp. 146–177, Wiley Online Library, 2013.

[5] R. Wang, J. Peethambaran, and D. Chen, "Lidar point clouds to 3-d urban models : a review," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 11, no. 2, pp. 606–627, 2018.

[6] Z. Kang, J. Yang, Z. Yang, and S. Cheng, "A review of techniques for 3d reconstruction of indoor environments," *ISPRS International Journal of Geo-Information*, vol. 9, no. 5, p. 330, 2020.

[7] O. Mattausch, D. Panozzo, C. Mura, O. Sorkine-Hornung, and R. Pajarola, "Object detection and classification from large-scale cluttered indoor scans," in *Computer Graphics Forum*, vol. 33, pp. 11–21, Wiley Online Library, 2014.

[8] Zynka BIM AB, 2014. https://bim.zynka.se/.

[9] V. Sanchez and A. Zakhor, "Planar 3d modeling of building interiors from point cloud data," in *2012 19th IEEE International Conference on Image Processing*, pp. 1777–1780, IEEE, 2012.

[10] R. Schnabel, R. Wahl, and R. Klein, "Efficient ransac for point-cloud shape detection," in *Computer graphics forum*, vol. 26, pp. 214–226, Wiley Online Library, 2007.

[11] L. Nan and P. Wonka, "Polyfit: Polygonal surface reconstruction from point clouds," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2353–2361, 2017.

[12] M. Li, P. Wonka, and L. Nan, "Manhattan-world urban reconstruction from point clouds," in *European Conference on Computer Vision*, pp. 54–69, Springer, 2016.

[13] F. Yang, G. Zhou, F. Su, X. Zuo, L. Tang, Y. Liang, H. Zhu, and L. Li, "Automatic indoor reconstruction from point clouds in multi-room environments with curved walls," *Sensors*, vol. 19, no. 17, p. 3798, 2019.

[14] J. Jung, C. Stachniss, and C. Kim, "Automatic room segmentation of 3d laser data using morphological processing," *ISPRS International Journal of Geo-Information*, vol. 6, no. 7, p. 206, 2017.

[15] C. Mura, O. Mattausch, A. J. Villanueva, E. Gobbetti, and R. Pajarola, "Automatic room detection and reconstruction in cluttered indoor environments with complex room layouts," *Computers & Graphics*, vol. 44, pp. 20–32, 2014.

[16] X.-F. Han, J. S. Jin, M.-J. Wang, W. Jiang, L. Gao, and L. Xiao, "A review of algorithms for filtering the 3d point cloud," *Signal Processing: Image Communication*, vol. 57, pp. 103–112, 2017.

[17] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[18] K. G. Derpanis, "Overview of the ransac algorithm," *Image Rochester NY*, vol. 4, no. 1, pp. 2–3, 2010.

[19] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *kdd*, vol. 96, pp. 226–231, 1996.

[20] Y. Cheng, "Mean shift, mode seeking, and clustering," *IEEE transactions on pattern analysis and machine intelligence*, vol. 17, no. 8, pp. 790–799, 1995.

[21] Design Council, "The 4d model or 'double diamond' design process model," 2012.

A

Evaluation, all runs

A.1 Big building

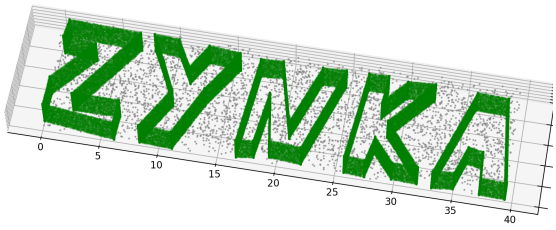
Name	Runs
Missing Area	[52.082, 56.4904, 24.8444, 57.4959, 24.2445, 61.9854]
Additional Area	[62.6424, 56.9524, 100.5526, 62.7004, 60.5703, 112.7447]
Additional Rooms	[21, 15, 18, 16, 25, 27]
Intersection Area	[1409.2163, 1405.6044, 1435.9195, 1401.6428, 1434.8023, 1397.7627]
Intersection Over Union	[0.8847, 0.8857, 0.8798, 0.88, 0.9034, 0.8514]
RMS distance to points	[0.0501, 0.0492, 0.054, 0.0492, 0.0494, 0.0514]
Run time (seconds)	[1402.6899, 1192.0026, 1402.5027, 1178.6827, 1431.7211, 1314.2809]
Union Area	[1592.8806, 1586.9675, 1632.0857, 1592.8091, 1588.2896, 1641.6778]
Number of Runs	6

A.2 Small building

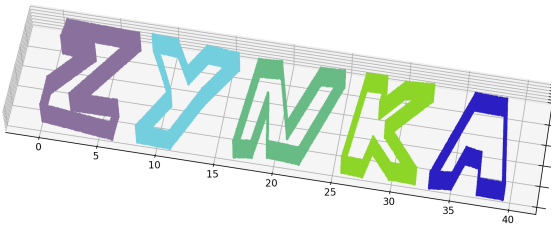
Name	Runs
Missing Area	[27.4586, 29.3266, 47.2273, 36.4611, 43.6646, 43.6031]
Additional Area	[54.2199, 55.4713, 60.1034, 54.4947, 61.5725, 49.044]
Additional Rooms	[18, 18, 20, 20, 14, 16]
Intersection Area	[469.596, 467.6841, 447.1707, 457.9721, 453.6574, 451.731]
Intersection Over Union	[0.8107, 0.8073, 0.7674, 0.7954, 0.7724, 0.793]
RMS distance to points	[0.0502, 0.0503, 0.0504, 0.0464, 0.0501, 0.049]
Run time (seconds)	[422.9979, 570.8253, 615.2898, 612.2862, 558.5564, 573.2421]
Union Area	[579.2194, 579.3493, 582.7312, 575.7534, 587.3002, 569.678]
Number of Runs	6

B

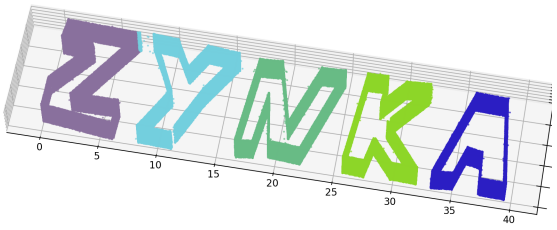
Synthetic evaluation, Zynka



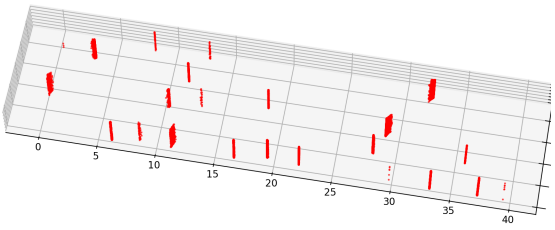
(a) Synthetic input



(b) Ground truth clusters



(c) Predicted clusters



(d) False negatives