Extraction and Merging of Topometric Maps

Maximiliaan van Schendel
student #4384644
m.vanschendel@tudelft.nl

1st supervisor: Edward Verbree
2nd supervisor: Pirouz Nourian
external supervisor: Robert Voûte

24/01/2022

Contents

abstract The merging of multiple partial maps of indoor environments created by teams of human or robot agents into a single global map is a key problem that, when solved, can enable co-localization and collaborative mapping of large environments. Existing map merging approaches generally depend on external signals which are not available indoors or only use the geometric properties of an environment. Inspired by the human understanding of environments in relationship to their context we propose a map merging system that extracts and uses topometric maps, a map representation containing both the geometrical and topological characteristics of an environments, to solve the map merging problem in indoor spaces. In this research we demonstrate an intuitive approach to extracting topometric maps of 3D, multi-floor, indoor environments. We then use both the topological and geometric characteristics contained in the topometric maps to perform context-aware map matching and fusion.

1 Introduction

3D maps of indoor environments are used for a wide variety of purposes ranging from vacuuming robots to industrial security. These maps are captured by a variety of agents, from humans using a 3D scanner to autonomous robots with integrated sensors. 3D maps can be used to provide comprehension of the environment to humans through cartography or for automated scene understanding that enables complex robot behaviour. In many cases it is advantageous for multiple mapping agents to collaboratively create a map, such as during rescue operations where the location of the subject(s) is unknown. By working together larger areas can be mapped in a shorter amount of time. Each agent may map the environment using a different approach, resulting in a more complete whole.

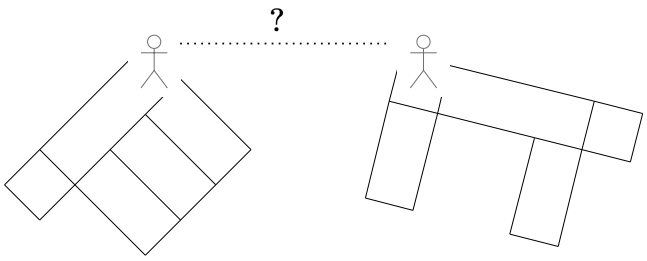

Figure 1: The map merging problem. How do you combine partial maps if relative positions are unknown?

The problem of creating a single, global map from multiple, partial maps is called map merging. Map merging is challenging in scenarios where the relative positions of agents are unknown because the relative positions could be used to align the maps directly. This is often the case in indoor environments because external positioning signals, e.g. GNSS are highly attenuated. This means that the global map must be created using only the properties of the partial maps. Map merging can be subdivided into two subproblems. The first is map matching, the identification of overlapping areas between partial maps. The second is map fusion, the alignment and merging of the partial maps based on the identified matches.

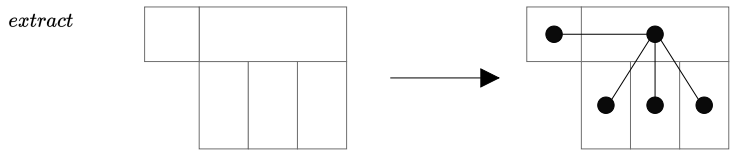

Figure 2: Extraction of topometric maps from raw data.

Map matching is essentially a problem of place recognition; identifying the same place in different maps. Human place recognition uses a combination of the characteristics of that place and its relationship to its context. To illustrate,

when asked to describe a room someone may answer that it has an L-shape, but also that the hallway leading upto it is shaped like a parallelogram or that the room has two opposing neighbouring rooms. A complete description of a place thus includes both its visible properties, such as its shape, and the topological structure of its environment. In this thesis we propose an approach to map matching that uses both of these aspects to identify matching rooms between partial maps. To achieve this we also propose an approach for extracting topometric maps, which represent both the environment's geometry and its topological structure, from point clouds. Finally, we also propose an approach to fuse the geometry and topology of topometric maps by using the identified matches.

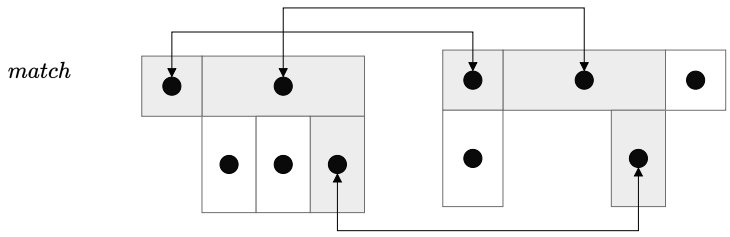

Figure 3: Map matching using both geometric and topological properties.

We hypothesize that the connectivity of places within the environment are consistently identifiable between partial maps. We further hypothesize that using the metric characteristics of the environment in conjunction with its topological characteristics will improve identification of overlapping areas over a purely geometric approach. Finally, we hypothesize that the identified matches can be used to fuse both the geometry and topology of the partial topometric maps into a global topometric map.

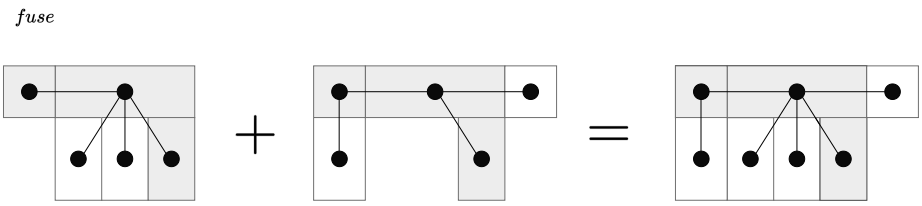

Figure 4: Fusion of partial topometric maps into a global topometric map.

The most important contributions of our work are:

1. Extraction of topometric maps from point clouds
2. Map matching using both the geometric and topological properties
3. Map fusion of topometric maps

2 Research Questions

2.1 Main question

How can we apply topometric representations of indoor environments to solve the map merging problem?

2.2 Subquestions

1. In what way can partial topometric maps be extracted from partial point cloud maps?
2. What approach is best suited for identifying matches between partial topometric maps?
3. How can the identified matches be used to fuse two or more partial topometric maps into a global topometric map?

2.3 Scope

During this thesis the following will be created.

1. A program that is able of topometric map extraction from point clouds and map merging.
2. An analysis of the program's performance on publically available standard datasets.
3. Reports containing documentation and background research.

To better delineate the scope of the thesis we provide several aspects that will **not** be researched or discussed.

1. Map merging using known relative poses between agents or meeting strategies. Agent behaviour is assumed to be independent and agents are not able to sense eachother.
2. Map merging using observations unrelated to the environment's geometrical and topological characteristics. E.g. the environment's colour or actively transmitted beacon signals.
3. Map merging assisted by a priori knowledge of the environment. E.g. building information models (BIM) or floor plans.
4. Map merging using the pose graphs of agents. Agent poses are assumed to be unknown.
5. Achieving (near) real-time performance.

3 Related work

3.1 Topometric map extraction

3.1.1 Voronoi graphs

3.1.2 Visibility clustering

3.2 Map matching

3.2.1 Topological matching

3.2.2 Spectral features

3.2.3 Deep features

3.3 Map fusion

3.3.1 Point cloud registration

3.3.2 Non-rigid registration

4 Methodology

In this section we will describe our methodology for solving the map merging problem. We divide our methodology into three major components: map extraction, map matching and map fusion, which correspond with the three subresearch questions. This section follows this division, with an added subsection describing the different map representations that we use. Figure 5 shows the steps of our methodology. Refer to the specific subsections for each step for a further description of the algorithms and notation. We will now give a short summary of each of the steps of our methodology.

Map extraction For each input partial map, in the form of a point cloud, we create a voxel grid with a given cell size. Within these voxel grids we detect which voxels could feasibly be used to navigate through the environment. Using this information we segment the voxel grids into submaps which closely match a human interpretation of how an indoor environment can be divided into rooms. We do so by finding areas with many common viewpoints. By combining the room submaps with the navigable voxels we can extract the environment's topological graph, which describes the adjacency relationships between rooms. We then create a topometric map for each input point cloud by merging the topological graph with the segmented voxel grid into a single map.

Map matching In the second part of our methodology, map matching, we identify matches between the rooms of the partial maps with the purpose of detecting overlapping areas. We do this by first generating a descriptor for each room that captures its geometric features and those of its context; the rooms that lie within a number of steps in the topological graph. We then find a matching by growing multiple matching hypotheses along the topological graphs in a constrained manner and selecting the one that contains the largest number of matches.

Map fusion In the final part of our methodology, map fusion, we find the transformation that aligns the partial maps' geometry and use it to create a single, global topometric map. We do so by finding the optimal transformation between each pair of matched rooms. We then cluster the transformations based on similarity and select the cluster whose mean transformation best aligns the partial maps as the most likely correct transformation. After using the mean transformation to align the geometry of the partial maps into a global voxel grid map we extract a topometric map from it to create the global topometric map.

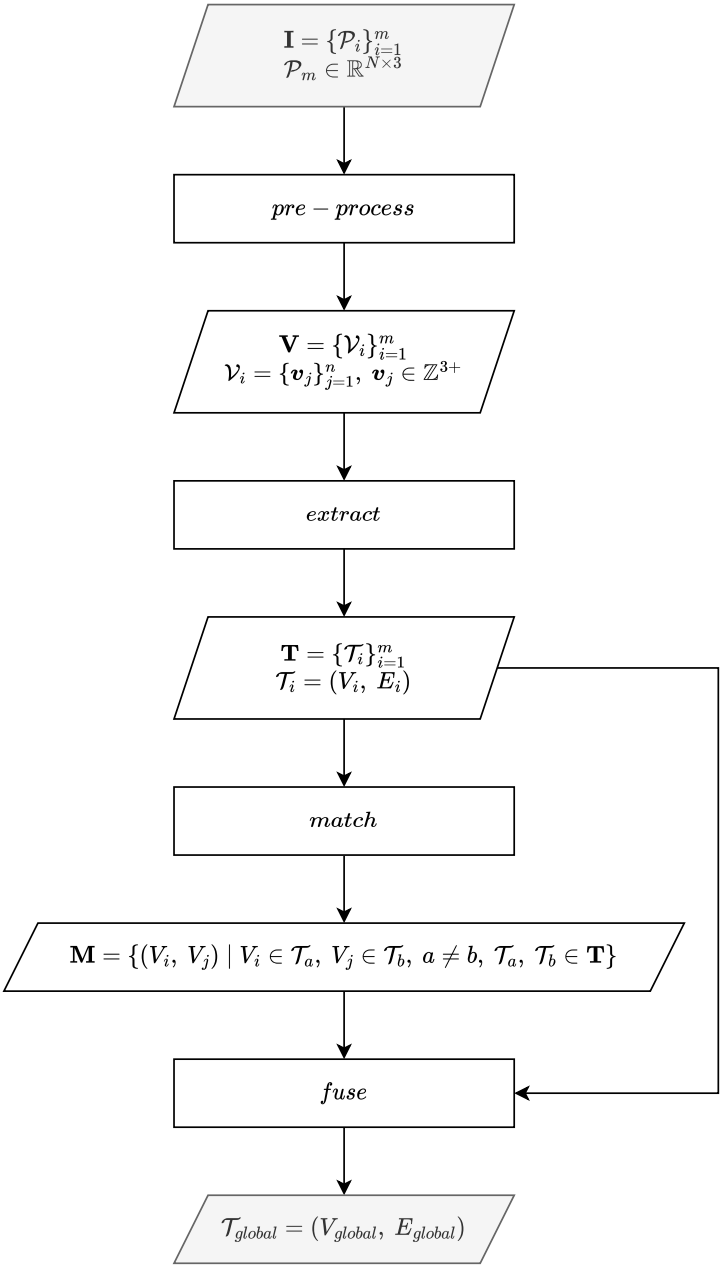

Figure 5: Diagram showing overview of methodology.

4.1 Map Representations

In this section we will give a description of the different kinds of map representation that are used in this research, their mathematical notation, and the operations that we perform on them.

4.1.1 Point Cloud

An unordered collection of points representing the geometry of an object or environment in 3D euclidean space (Volodine, 2007).

$$\mathcal{P} = \{p_i\}_{i=1}^{n}, p_i \in \mathbb{R}^3 \tag{1}$$

Where $\mathcal{P}$ denotes the point cloud and n the number of points that it contains.

4.1.2 Voxel Grid

A voxel is the 3D equivalent of a pixel. A voxel represents a single cell in a bounded 3D volume divided into a regular voxel grid. A voxel may contain information about whether it is occupied, what color it is, or any other property. A voxel can be represented by a three-dimensional vector containing its coordinates along the x, y and z axes of the voxel grid, as shown in equation 2.

$$\boldsymbol{v} = (x, y, z) \in \mathbb{Z}^{3+} \tag{2}$$

We define a voxel grid as a set of voxels with an associated edge length e_l, as shown in equation 3. Figure 6 shows an example voxel grid and its components.

$$\mathcal{V} = \{\boldsymbol{v}_i\}_{i=1}^{n},\ n \in [1, \prod \boldsymbol{v}_{\boldsymbol{V_{max}}}] \tag{3}$$

To generate a voxel grid we divide a 3D axis-aligned volume V, defined by minimum and maximum bounds $V_{min}, V_{max} \in \mathbb{R}^3$ into a grid of cubic cells with edges of length e_l. A voxel represents a subvolume of V bounded by a single cell. A voxel coordinate only consists of integer values that represent multiples of the edge length along each dimension. Voxel $\boldsymbol{v}_{\boldsymbol{V_{min}}} = (0, 0, 0)$ represents the first cell along each of the voxel grid's axes and the minimum of the volume's bounds, voxel represents $(0, 1, 1)$ the first cell along the x and the second along the y and z axes, etc. We also restrict voxel coordinates to only be positive as negative coordinates would fall outside of the bounds of the volume. For the same reason a voxel's coordinates can not be larger than that of the voxel representing the volume's maximum bounds $\boldsymbol{v}_{\boldsymbol{V_{max}}} = (V_{max} - V_{min})\ //\ e_l$, where // denotes integer division.

$$\boldsymbol{v}_{min} = V_{min} + \boldsymbol{v} * e_l \tag{4}$$

$$\boldsymbol{v}_{max} = \boldsymbol{v}_{min} + e_l \tag{5}$$

The minimum and maximum bounds of this subvolume are given by equation 4 and 5. The voxel's centroid is given by equation 6. Given a point $\boldsymbol{p}$ within the bounds of V, the corresponding voxel is given by equation 7 .

$$\boldsymbol{v_c} = (\boldsymbol{v_{min}} + \boldsymbol{v_{max}}) * 0.5 \tag{6}$$

$$\boldsymbol{v_p} = (\boldsymbol{p} - V_{min}) \ // \ e_l \tag{7}$$

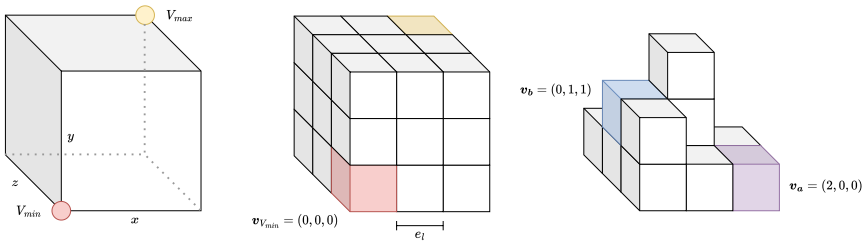

Figure 6: A voxel grid and its components.

A property of a voxel is given by equation 8. We define a property as an m by n real matrix for the purposes of this research. However, in reality a property could be any value. For example, we denote the property *occupied*, which can be either 1 or 0, of a voxel $\boldsymbol{v}$ as $V_{occupied}(\boldsymbol{v}) = \boldsymbol{v}_{occupied} = \big(1\big) \,|\, \big(0\big)$

$$\mathcal{V}_{property} : \mathbb{Z}^{3+} \mapsto \mathbb{R}^{m*n}, \ \mathcal{V}_{property}(\boldsymbol{v}) = \boldsymbol{v}_{property} \tag{8}$$

Sparse Voxel Octree Several operations on voxel grids benefit from using a spatial index, including radius searching and level of detail generation. We use a data structure called a sparse voxel octree (SVO) to achieve this. A normal octree recursively subdivides a volume into 8 cells, called octants. This operation results in a tree data structure, with nodes representing octants at a certain level of subsidivision. The root node of the tree structure represents the entire volume while the leaf nodes represent batches of 1 or more data points. In the case of a sparse voxel octree the leaf nodes represent individual voxels, with only the octants containing an occupied voxel represented in the tree.

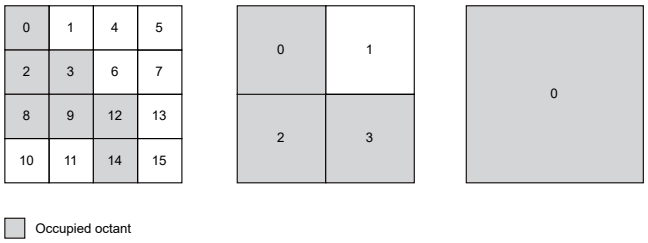

Figure 7: 2D example of morton codes.

To generate the SVO we first create a Morton order for the voxel grid. A Morton order maps the three-dimensional coordinates of the voxels to one dimension while preserving locality. It does by interleaving the binary representation of the voxel's coordinates into a single binary string which is interpreted as a positive integer, a Morton code. The ascending sorted vector of Morton codes gives us the Morton order. We divide the Morton order into buckets with size 8, such that each bucket contains at most 8 Morton codes, with a maximum difference of 8. Each non-empty bucket represents a parent node of at most 8 child nodes in the octree. By recursively performing this step until only one bucket remains, the root node, we can construct an SVO.

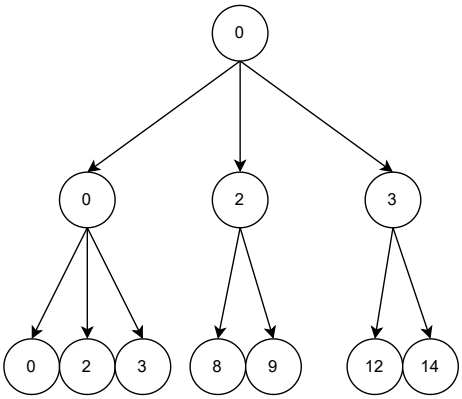

Figure 8: Example of a sparse voxel octree generated from the above Morton codes.

We denote the function that returns all n voxels within range r of a voxel as follows.

$$radius : \mathbb{Z}^{3+},\ \mathbb{R} \mapsto \mathbb{Z}^{n\times 3} \tag{9}$$

The voxels within a sphere around a point can be found by recursively intersecting the sphere with the octants of the SVO. If the sphere does not intersect with an octant then none of its leaf nodes do and the corresponding voxels are not within the sphere. If an octant does intersect with the sphere then its children are tested for intersection. The algorithm returns all leaf nodes that intersect with the sphere.

Voxel convolution Voxel convolution involves moving a sliding window, or kernel, over each voxel in the grid to retrieve its neighbourhood and then computing a new value for the voxel based on the weighted sum of its neighbours. We can define a kernel $\mathcal{K}$ as a voxel grid with an associated weight for each voxel and an origin voxel.

$$weight : \mathbb{Z}^{3+} \mapsto \mathbb{R} \tag{10}$$

$$\boldsymbol{o}_{\mathcal{K}} \in \mathbb{Z}^3 \tag{11}$$

To apply a kernel to a voxel we first translate the kernel so that its origin lies on the voxel.

$$\mathcal{K}_t = \{\boldsymbol{v}_{\boldsymbol{\mathcal{K}}} + (\boldsymbol{v} - \boldsymbol{o}_{\boldsymbol{\mathcal{K}}}) \mid \boldsymbol{v}_{\boldsymbol{\mathcal{K}}} \in \mathcal{K}\} \tag{12}$$

We then get the property which we wish to convolve of each neighbour, multiply it by the neighbour's weight and sum it.

$$\mathcal{K}_{property}(\boldsymbol{v}) = \sum \{weight(\boldsymbol{v})\mathcal{V}_{property}(\boldsymbol{v}) \mid \boldsymbol{v} \in \mathcal{K}_t \cap \mathcal{V}\} \tag{13}$$

We denote the convolution of a property of every voxel in $\mathcal{V}$ with $\mathcal{K}$ as follows.

$$\mathcal{V}_{property,\ \mathcal{K}} = \mathcal{V} * \mathcal{K}_{property} = \{\mathcal{K}_{property}(\boldsymbol{v}) \mid \boldsymbol{v} \in \mathcal{V}\} \tag{14}$$

Neighbourhood graph The neighbourhood graph of a voxel grid represents the connectivity between voxels as undirected, unweighted graph. The nodes of the neighbourhood graph correspond to individual voxels and the edges to whether two voxels can be considered neighbours. Whether two voxels are neighbours is defined by a kernel which has only 1 or 0-valued weights. If, when applying the kernel to a voxel, another voxel within that kernel is occupied and the kernel's weight for that position is 1 then the two voxels are neighbours. The neighbourhood graph allows us to perform graph operations, such as identifying connected components, on voxel grids. Figure 9 shows two commonly used kernels for constructing neighbourhood graphs, the Von Neumann and Moore neighbourhoods, also respectively known as the 6-neighbourhood and the 26-neighbourhood.

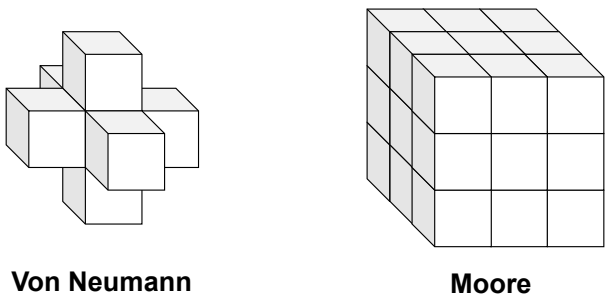

Figure 9: Two common connectivity kernels. Note that the center voxel's weight is 0, as a voxel does not neighbour with itself.

4.1.3 Topological map

Topological maps are a graph representation of an environment's structure, where vertices represent locally distinctive places, often rooms, and edges represent traversable paths between them (see figure **??**) (Thrun, 1998; Kuipers & Byun, 1988). Topological maps are based on observations that cognitive maps, the mental maps used by humans to navigate within an environment, consist of multiple layers with a topological description of the environment being a fundamental component (Kuipers & Byun, 1988; Kuipers, 1978).

We denote a topological map as shown in equations 15, 16 and 17. The topological map consists of a graph G, where nodes N represent distinctive places n and edges E represent the presence of a navigable path between neighbouring pairs of places (v_j, v_k) that does not pass through any other places. Whether a path is navigable depends on who or what is navigating. For the purpose of this thesis a navigable path is a path that can be reasonably used by humans to walk from one room to another. Following this definition, only a part of the environment can be used as a navigable path. This includes the parts of the floor that are at sufficient distance from a wall or ceiling, stairs, and ramps.

$$G = (N, E) \tag{15}$$

$$N = \{n_i\}_{i=1}^{k} \tag{16}$$

$$E = \{(n_j, n_k)_i\}_{i=1}^{m},\ n_j \in N,\ n_k \in N,\ n_j \neq n_k \tag{17}$$

Figure 10 shows an example topological map of a house with five rooms and their connectivity.

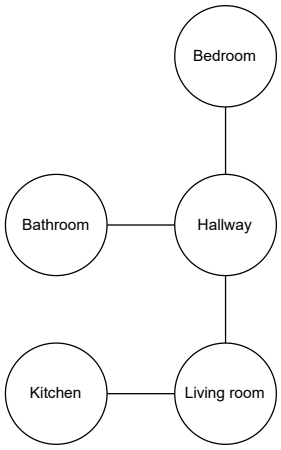

Figure 10: Example of a topological map.

4.1.4 Topometric map

A hybrid map representation combining both the topological and geometric characteristics of the environment. This map representation allows the end-user to use either topological or metric information depending on the needs of the situation, e.g. the topological layer can be used for large-scale navigation and abstract reasoning while the metric layer can be used for place recognition and obstacle avoidance. In the context of this thesis a topometric map refers to a graph representation of an indoor environment where the nodes represent rooms and their associated geometry as a voxel grid and the edges represent the navigability relationship between them. It is thus a hybrid representation of the environment that combines the properties of the voxel grid and the topological map which we described above.We denote a topometric map $\mathcal{T}$ as shown in equation 18, where $\mathcal{V}$ represents the complete geometry of the environment and G the topological graph.

$$\mathcal{T} = (\mathcal{V},\ G),\ \mathcal{V} = \{\boldsymbol{v}_i\}_{i=1}^{n} \tag{18}$$

The topological graph, which we denote as in 19, consists of a set of nodes N and a set of edges E. Each node $n \in N$ represents a room and contains a subset of $\mathcal{V}$ that describes the geometry of that room. The nodes' subsets of $\mathcal{V}$ are not allowed to overlap, which means they represent a segmentation of $\mathcal{V}$.

$$G = (N,\ E),\ N = \{n_i\}_{i=1}^{k},\ n \subset \mathcal{V} \tag{19}$$

Figure 11 shows an example of a topometric map of an indoor environment.

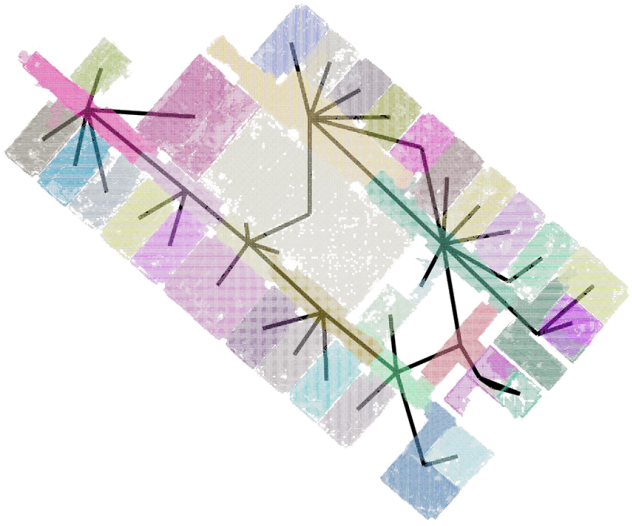

Figure 11: Example of a topometric map.

4.2 Map Extraction

The first step of our approach is topometric map extraction. The goal of this step is to transform a partial voxel grid map of an indoor environment, denoted by $\mathcal{V}$, into a topometric map, denoted by $\mathcal{T}$. In this section we propose an algorithm to achieve this goal. In an overview, it works as follows.

4.2.1 Overview

We first extract a navigation graph $\mathcal{G}_{navigation}$, the neighbourhood graph of all voxels which a hypothetical human agent could use to move through the environment, from $\mathcal{V}$. Using $\mathcal{G}_{navigation}$ we compute points where the hypothetical agent would have an optimal view of the environment. We then find the visible voxels for each of these points. By clustering the resultant visibilities based on similarity we segment $\mathcal{V}$ into submaps that align closely with how humans may divide indoor environments into rooms. As such, we refer to the submaps of $\mathcal{V}$ when segmented using visibility clustering as 'rooms'. We then construct the topological graph of the environment by finding which rooms have adjacent voxels in $\mathcal{G}_{navigation}$. Finally, we fuse the topological graph with the segmented map to construct the topometric map $\mathcal{T}$.

Figure 12 shows an overview of our map extraction algorithm, its input, and intermediate outputs. In the rest of this subsection we will discuss the algorithm in more detail.

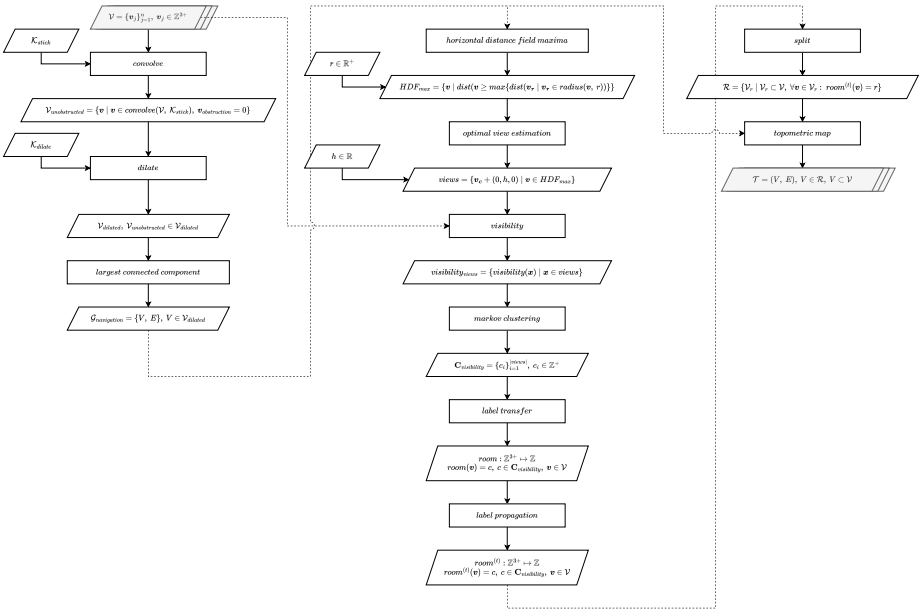

Figure 12: Diagram showing map extraction processes and intermediate data.

4.2.2 Navigation graph

In our first step we extract a navigation graph $\mathcal{G}_{navigation}$. The navigation graph is a connected graph which tells us how a theoretical agent in the map's environment would navigate through that environment. In practice, assuming a human agent, this means the areas of the floor, ramps and stairs that are at a sufficient distance from a wall and a ceiling. We compute $\mathcal{G}_{navigation}$ using a three step algorithm which we describe below.

Convolution The first step of navigation graph extraction uses voxel convolution with a stick-shaped kernel $\mathcal{K}_{stick}$ (shown in figure 13) to find all voxels that are unobstructed and may thus be used to navigate the environment. Each voxel in the kernel has a weight of 1, except the origin voxel which has weight 0. Convolving the voxel grid's occupancy property with the stick kernel gives us each voxel's obstruction property, with a value of 0 when no other voxels are present in the stick kernel. This indicates that these voxels have enough space around and above them to be used for navigation. We then filter out all obstructed voxels, leaving only the voxels that could be used for navigation.

$$\mathcal{V}_{unobstructed} = \{\boldsymbol{v} \mid \boldsymbol{v} \in \mathcal{V} * \mathcal{K}_{stick},\ \mathcal{K}_{stick}(\boldsymbol{v}) = 0\} \tag{20}$$

Upwards dilation The next step of the algorithm is to dilate the unobstructed voxels upwards by a distance d_{dilate}. This connects the voxels separated by a small height differences into a connected volume, which is necessary for the navigation graph to be able to connected stairs. The value of d_{dilate} depends on the expected differences in height between the steps of stairs in the environment. Typically we use a value of 0.2m for this parameter. The result is a new voxel grid $\mathcal{V}_{dilated}$. Note that the dilation step may create new occupied voxels that are not in the original voxel grid, which means that $\mathcal{V}_{dilated}$ is not a subset of $\mathcal{V}$.

Connected components The final step of the algorithm is to split $\mathcal{V}_{dilated}$ into one or more connected components. A connected component $\mathcal{V}_i$ of a voxel grid is a subset of $\mathcal{V}$ where there exists a path between every voxel in $\mathcal{V}_i$. We denote the set of all connected components as $\mathcal{C} = \{\mathcal{V}_i\}_{i=1}^{n}$. To find the connected components we first find the neighbourhood graph of $\mathcal{V}_{dilated}$ using the Von Neumann neighbourhood kernel $\mathcal{K}_6$, which is shown in equation 21.

$$\mathcal{G}_{\mathcal{K}_6} = (N,\ E),\ N = \mathcal{V}_{dilated} \tag{21}$$

We then find the connected components of the neighbourhood graph using algorithm 0. After doing so, we find the connected component with the largest amount of voxels and use it as the navigation graph $\mathcal{G}_{navigation}$. We denote the intersection of the voxels in the navigation graph with $\mathcal{V}$ as $\mathcal{V}_{navigation}$.

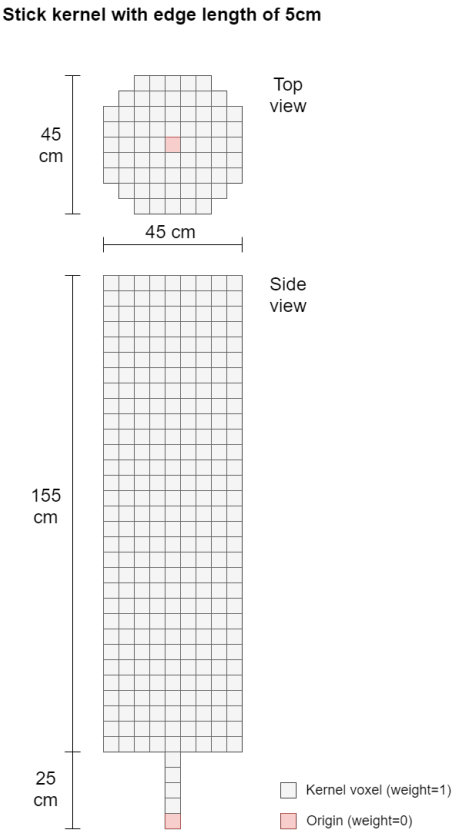

Figure 13: Illustration of stick kernel with top and side views.

4.2.3 Room segmentation

The next step of our approach is to segment the complete voxel grid map $\mathcal{V}$ into non-overlapping rooms. We do so by using a visibility clustering approach. We will now describe the algorithm that we use to achieve this.

Maximum visibility estimation To segment the map into rooms using visibility clustering it is first necessary to identify the viewpoints that will be used to compute the visibilities. Ideally, we want viewpoints that maximize the view of the environment. We find these viewpoints by finding the points that are at a maximum distance from the boundary of the navigation graph within their local neighbourhood. The reasoning behind this is that the points that maximize the view of the environment should be equally spaced and as far away from any obstruction as possible. We compute these points as follows.

For each voxel in the navigation graph we compute the horizontal Manhattan distance to the nearest boundary voxel. A boundary voxel is a voxel for which

Algorithm 1 Region growing connected components

Input Dilated voxel grid $\mathcal{V}_{dilated}$
Input Von Neumann Connectivity kernel $\mathcal{K}_6$
Output Connected components $\mathcal{C}$

$\mathcal{G}_{\mathcal{K}_6} = (N,\ E),\ N = \mathcal{V}_{dilated}$ ▷ Convert voxel grid to neighbourhood graph
$N_{unvisited} = N$
$\mathcal{C} = \{\}$
while $|N_{unvisited}| \neq 0$ **do**
  Select random node n from $N_{unvisited}$
  Remove $BFS(n)$ from $N_{unvisited}$ ▷ Breadth-first search to find connected nodes
  Add $BFS(n)$ to $\mathcal{C}$
end while

not every voxel in its Von Neumann neighbourhood is occupied. To compute this value we iteratively convolve the voxel grid with a circle-shaped kernel on the X-Z plane, where the radius of the circle is expanded by 1 voxel with each iteration, starting with a radius of 1. When the number of voxel neighbours within the kernel is less than the number of voxels in the kernel a boundary voxel has been reached. The number of radius expansions tells us the Manhattan distance to the boundary of a particular voxel. We denote the horizontal distance of a voxel to its boundary as $dist : \mathbb{Z}^{3+} \mapsto \mathbb{Z}$. Computing the horizontal distance for every voxel in $\mathcal{V}$ gives us the horizontal distance field (HDF), such that

$$HDF = \{dist(\boldsymbol{v}) \mid \boldsymbol{v} \in \mathcal{V}\} \tag{22}$$

$$HDF_{max} = \{\boldsymbol{v} \mid dist(\boldsymbol{v}) \geq max\{dist(\boldsymbol{v_r} \mid \boldsymbol{v_r} \in radius(\boldsymbol{v},\ r))\}\} \tag{23}$$

We denote the horizontal distance of a given voxel $\boldsymbol{v}$ as $d_{\boldsymbol{v}}$. We implement this using the following algorithm.

Algorithm 2 Horizontal distance field

Input Navigation voxel grid $\mathcal{V}_{navigation}$
Output Horizontal distance field $HDF = \{dist(\boldsymbol{v}) \mid \boldsymbol{v} \in \mathcal{V}_{navigation}\}$

for each $\boldsymbol{v} \in \mathcal{V}_{navigation}$ **do**
  $r = 1$
  Create $\mathcal{K}_{circle}$ with radius r
  while $\mathcal{K}_{circle}(\boldsymbol{v}) = |\mathcal{K}_{circle}|)$ **do**
    $r = r + 1$
    Expand $\mathcal{K}_{circle}$ with new radius r
  end while
  $HDF = HDF \cup r$ ▷ Add voxel's radius to horizontal distance field
end for

We then find the maxima of the horizontal distance field within a given radius $r \in \mathbb{R}$. The local maxima of the horizontal distance field are all voxels that have a larger or equal horizontal distance than all voxels within r, such that equation 23 follows. Increasing the value of r reduces the number of local maxima and vice versa. All voxels in HDF_{max} lie within the geometry of the environment, which means the view of the environment is blocked by the surrounding voxels. To solve this, we take the centroids of the voxels in HDF_{max} and translate them upwards to a reasonable scanning height h. We denote these positions as

$$views = \{\boldsymbol{v_c} + (0, h, 0) \mid \boldsymbol{v} \in HDF_{max}\} \tag{24}$$

Figure 14 shows an illustration of the horizontal distance field computation and the identification of its local maxima. Figure 15 shows a real example of the horizontal distance field extracted from a small two-storey environment in grayscale and the associated *views* in red.

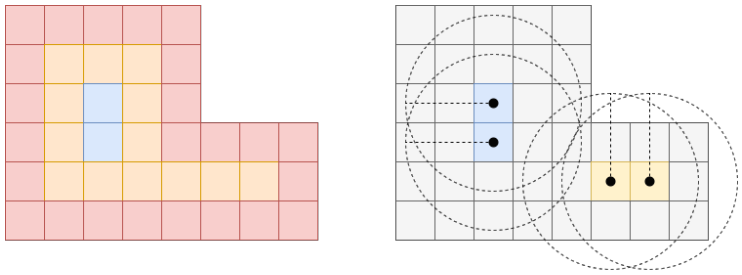

Figure 14: Illustration of horizontal distance field computation and extraction of local maxima.

Visibility computation The next step in the room segmentation algorithm is to compute the visibility from each position in *views*. We denote the all voxels that are visible from a given position as

$$visibility : \mathbb{R},\ \mathbb{Z}^{n\times 3} \mapsto \mathbb{Z}^{m\times 3},\ m \in \mathbb{R},\ n \geq m \tag{25}$$

A target voxel is visible from a position if a ray cast from the position towards the centroid of the voxel does not intersect with any other voxel. To compute this we use the digital differential analyzer (DDA) algorithm to rasterize the ray onto the voxel grid in 3D (see algorithm 0). We then check if any of the voxels that the ray enters- except the target voxel- is occupied. If none are, the target voxel is visible from the point. Figure 16 shows a 2D representation of how the DDA algorithm works.

We perform this raycasting operation from every position in *views* towards every voxel within a radius r_v of that position. Only taking into account voxels

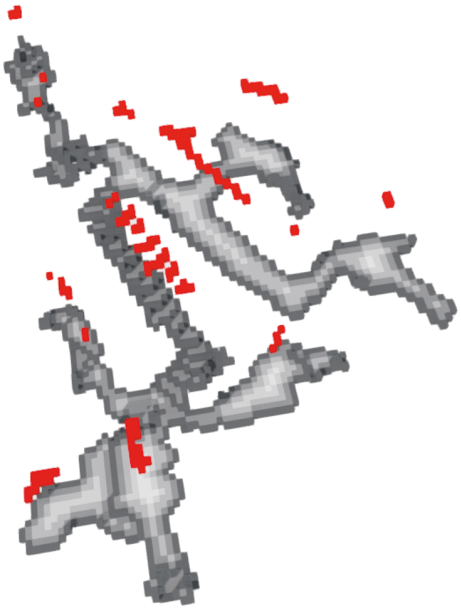

Figure 15: Resulting horizontal distance field of partial map, with resultant optimal view points shown in red.

within a radius speeds up the visibility computation, and is justifiable based on the fact that real-world 3D scanners have limited range. We denote the set of visibilities from each point in views as

$$visibility_{views} = \{visibility(\boldsymbol{x}) \mid \boldsymbol{x} \in views\} \tag{26}$$

$$visibility(\boldsymbol{o}) = \{\boldsymbol{v} \mid \boldsymbol{v} \in radius(o, r_v) \wedge DDA(\mathcal{V}, \boldsymbol{o}, \boldsymbol{v}) = \boldsymbol{v}\} \tag{27}$$

Visibility clustering After computing the set of visibilities from the estimated optimal views we apply clustering to group the visibilities by similarity. This is based on the definition of a room as a region of similar visibility. Remember that each visibility is a subset of the voxel grid map. To compute the similarity of two sets we use the Jaccard index, which is given by equation 28.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{28}$$

Computing the Jaccard index for every combination of visibilities gives us a similarity matrix $S^{n \times n} \in [0, 1]$. An example similarity matrix is shown in figure 17.

Algorithm 3 DDA (Digital Differential Analyzer)

Input Voxel grid $\mathcal{V}$
Input Ray origin $\boldsymbol{o} \in \mathbf{R}^3, \boldsymbol{o} \in [\mathcal{V}_{min}, \mathcal{V}_{max}]$
Input Ray target $\boldsymbol{t} \in \mathbf{R}^3$
Output $hit \in \mathbb{Z}^3$ ▷ First encountered collision

$\boldsymbol{p}_{current} = \boldsymbol{o}$
$\boldsymbol{v}_{\boldsymbol{o}} = (\boldsymbol{o} - \mathcal{V}_{min})//\mathcal{V}_e$
$\boldsymbol{v}_{current} = \boldsymbol{v}_{\boldsymbol{o}}$
$\boldsymbol{d} = (\boldsymbol{t} - \boldsymbol{o})$
$heading = \boldsymbol{d} \odot abs(\boldsymbol{d})^{-1}$ ▷ Determine if ray points in positive or negative direction for every axis
while $(\boldsymbol{v}_{current} \notin \mathcal{V} \vee \boldsymbol{v}_{current} = \boldsymbol{v}_{\boldsymbol{o}}) \wedge \boldsymbol{p}_{current} \in [\mathcal{V}_{min}, \mathcal{V}_{max}]$ **do**
    $\boldsymbol{c} = centroid(\boldsymbol{v}_{current})$
    $d_{planes} = \boldsymbol{c} + heading * \mathcal{V}_e/2$
    $d_{min} = \infty$
    $axis = 1$
    for each (**do**$d \in d_{planes}$)
        $t = \frac{d - \boldsymbol{n} \cdot \boldsymbol{p}_{current}}{\boldsymbol{n} \cdot (\boldsymbol{t} - \boldsymbol{p}_{current})}$
        $\boldsymbol{i} = \boldsymbol{p}_{current} + t(\boldsymbol{t} - \boldsymbol{o})$
        if $d_{min} \geq t$ **then**
            $\boldsymbol{p}_{current} = \boldsymbol{i}$
            $\boldsymbol{v}_{current,\ axis} += heading_{axis}$
        end if
        $axis = axis + 1$
    end for
    $hit = \boldsymbol{v}_{current}$
end while

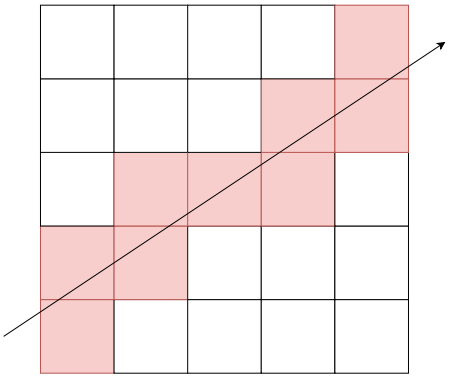

Figure 16: 2D representation of voxel raycasting.

We can also consider $S^{n\times n}$ as an undirected weighted graph $\mathcal{G}_S$, where every node represents a visibility and the edges the Jaccard index of two visibilities, as illustrated in figure 18.

This means we can treat visibility clustering as a weighted graph clustering problem. To solve this problem we used the Markov Cluster (MCL) algorithm, which has been shown by previous research to be state-of-the-art for visibility clustering. The main parameter of the MCL algorithm is inflation. By varying this parameter between an approximate range of $[1.2, 2.5]$ we get different clustering results. We find the optimal value for inflation within this range by maximizing the clustering's modularity. This value indicates the difference between the fraction of edges within a given cluster and the expected number of edges for that cluster if edges are randomly distributed. We denote the clustering of $visibility_{views}$ that results from the MCL algorithm as

$$\mathbf{C}_{visibility} = \{c_i\}_{i=1}^{|views|},\ c_i \in \mathbb{Z}^+ \tag{29}$$

Where the ith element of $\mathbf{C}_{visibility}$ is the cluster that the ith element of $visibility_{views}$ belongs to, such that for a given value of c, the elements in $visibility_{views}$ for which the corresponding c in $\mathbf{C}_{visibility}$ has the same value belong to the same cluster. As each visibility is a subset of the map, each cluster of visibilities is also a subset of the map. We denote the union of the visibilities belonging to each cluster as $\mathcal{V}_c$.

Label propagation It is possible for visibility clusters in $\mathcal{V}_c$ to have overlapping voxels. This means that each voxel in the partial map may have multiple associated visibility clusters. However, the goal is to assign a single cluster to each voxel in the map to create a non-overlapping segmentation. To solve this we assign to each voxel the cluster which contains the most visibilities that include that voxel. The result is a mapping from voxels to visibility clusters, which we will from now on refer to as rooms, as shown in equations 30 and 31.

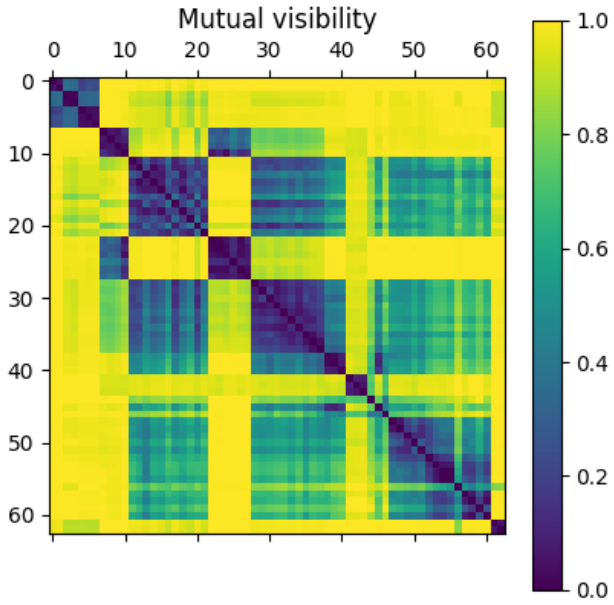

Figure 17: Similarity matrix extracted from set of visibilities. Each value represents the Jaccard index of two visibilities.

$$room : \boldsymbol{v} \mapsto \mathbb{Z} \tag{30}$$

$$room(\boldsymbol{v}) = c,\ c \in \mathbf{C}_{visibility},\ \boldsymbol{v} \in \mathcal{V} \tag{31}$$

This often results in noisy results, with small, disconnected islands of rooms surrounded by other rooms. Intuitively, this does not correspond to a reasonable room segmentation. To solve this we apply a label propagation algorithm. This means that for every voxel we find the voxels within a neighbourhood as defined by a convolution kernel. We then assign to the voxel the most common label, in this case the room, of its neighbourhood, if that label is more common than the current label. We iteratively apply this step until the assigned labels stop changing. Depending on the size of the convolution kernel the results are smoothed and small islands are absorbed by the surrounding rooms. Algorithm 0 shows how label propagation works.

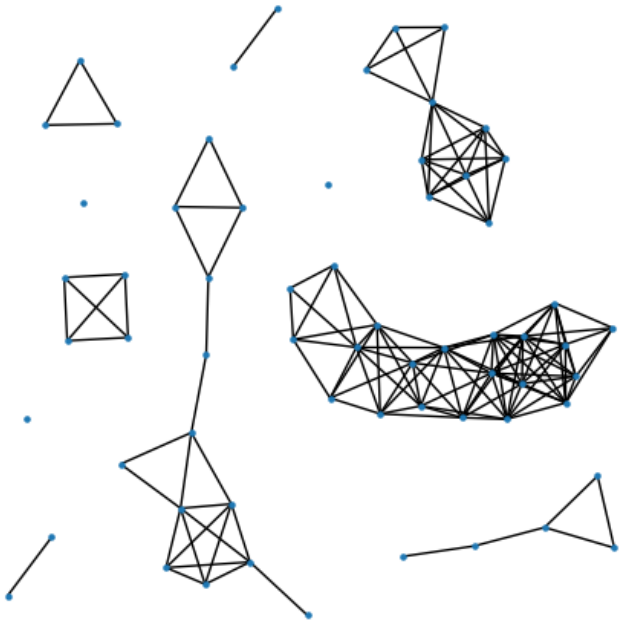

Figure 18: Graph representation of the similarity matrix, edges under a threshold similarity are removed. Nodes represent visibilities.

4.2.4 Topometric map extraction

The above steps segment the map into multiple non-overlapping rooms based on visibility clustering. In the next step we transform the map into a topometric representation $\mathcal{T} = (\mathcal{G}, \mathcal{V})$, which consists of a topological graph $\mathcal{G} = (N, E)$ and a voxel grid map $\mathcal{V}$. Each node in $\mathcal{G}$ represents a room, and also has an associated voxel grid which is a subset of $\mathcal{V}$ and represents the geometry of that room. Edges in $\mathcal{G}$ represent navigability between rooms, meaning that there is a path between them on the navigable volume that does not pass through any other rooms. This means that for two rooms to have a navigable relationship they need to have adjacent voxels that are both in the navigable volume. To construct the topometric map we thus add a node for every room in the segmented map with its associated voxels, we then add edges between every pair of nodes that satisfy the above navigability requirement. The exact method for extracting a topometric map from a room segmentations and the navigable voxels is shown in algorithm 0

Algorithm 4 Label propagation

Input Voxel grid $\mathcal{V}$
Input Initial labeling $label^{(0)} : \mathbb{Z}^{3+} \mapsto \mathbb{Z}$
Input Kernel $\mathcal{K}$
Output Propagated labeling after t steps $label^{(t)} : \mathbb{Z}^{3+} \mapsto \mathbb{Z}$

$t = 0$
while (**do**$label^{(t)} \neq label^{(t+1)}$) ▷ Keep iterating until labels stop changing
  for each (**do**$\boldsymbol{v} \in \mathcal{V}$)
    $L = \{label^{(t)}(\boldsymbol{v}_{nb}) \mid \boldsymbol{v}_{nb} \in neighbours(\boldsymbol{v}, \mathcal{K})\}4$
    $l_{max} = argmax_l \ |\{l \mid l \in L\}|$ ▷ Most common label in neighbourhood
    $l_{current} = label^{(t)}(\boldsymbol{v})$ ▷ Label of current voxel
    if $|\{l \mid l \in L \wedge l = l_{max}\}| > |\{l \mid l \in L \wedge l = l_{current}|$ **then**
      $label^{(t+1)}(\boldsymbol{v}) = l_{max}$
    else
      $label^{(t+1)}(\boldsymbol{v}) = label^{(t)}(\boldsymbol{v})$
    end if
  end for
  $t = t + 1$ ▷ Use propagated labeling as input for next iteration
end while

Algorithm 5 Topometric map extraction

Input Voxel grid $\mathcal{V}$
Input Voxel grid navigation subset $\mathcal{V}_{navigation}$
Input Room segmentation $room : \boldsymbol{v} \mapsto \mathbb{Z}$
Output Topological graph $\widetilde{\mathcal{G}}_{topology} = (V_{topology},\ E_{topology})$

$\widetilde{\mathcal{G}}_{topology} = (\{\},\ \{\})$
Get each unique room label $\mathbf{R} = \{room(\boldsymbol{v}) \mid \boldsymbol{v} \in \mathcal{V}\}$
Split $\mathcal{V}$ by label, such that $\mathbf{V} = \{\mathcal{V} \cap \{\boldsymbol{v} \mid \boldsymbol{v} \in \mathcal{V} \wedge room(\boldsymbol{v}) = r\} \mid r \in \mathbf{R}\}$
Store room label associated with each voxel grid $room_{\mathcal{V}} : r \mapsto \mathcal{V}$
$V_{topology} = \mathbf{V}$
for each (**do**$\boldsymbol{v} \in \mathcal{V}_{navigation}$)
  $v_r = room(\boldsymbol{v})$
  $nbs_r = \{room(nb) \mid nb \in neighbourhood(\boldsymbol{v},\ \mathcal{K}_{adjacency})\}$
  $r_{adjacency} = \{(r_a,\ r_b) \mid (r_a,\ r_b) \in v_r \times nbs_r \wedge r_a \neq r_b\}$
  $E_{topology} = E_{topology} \cup \{(room_{\mathcal{V}}(r_a),\ room_{\mathcal{V}}(r_b)) \mid (r_a,\ r_b) \in r_{adjacency}\}$
end for

4.3 Map Matching

4.3.1 Overview

The process of identifying overlapping areas between partial maps is called map matching. In the case of topometric map matching, this refers to identifying which nodes represent the same rooms between two partial maps. We denote our two partial topometric maps as

$$\mathcal{T}_a = (\mathcal{G}_a,\ \mathcal{V}_a),\ \mathcal{G}_a = (N_a, E_a) \tag{32}$$

$$\mathcal{T}_b = (\mathcal{G}_b,\ \mathcal{V}_b),\ \mathcal{G}_b = (N_b, E_b) \tag{33}$$

The goal of map matching is to find a one-to-one mapping between the rooms of both partial maps which corresponds to the real world and is robust to differences in coordinate system, resolution and quality between partial maps.

To identify matches between rooms we need to be able to compute their similarity. To do so, we first transform each room into a descriptor, an n-dimensional vector, which represents both the geometry of the room. The descriptor of two nodes with similar geometry should be close to eachother in feature space, meaning the distance between their vectors should be small. Conversely, the descriptors of two dissimilar rooms should be far away from eachother in feature space.

We then use the topological properties of the topometric maps to improve map matching in two ways. The first is contextual embedding. This means that we combine the descriptor of each room with the descriptor of its neighbourhood in the topological graph. This improves matching because multiple rooms may have similar geometry but not necessarily similar neighbourhoods. The second is hypothesis growing, which means that we grow multiple matching hypotheses along the topological graph in a constrained manner and only use the hypothesis that contains the most matches.

Figure 19 shows an overview of the steps described above. In the rest of this section we will describe the aforementioned steps in depth.

4.3.2 Geometric descriptor

Geometric feature embedding transforms a geometric object, in our case a voxel grid, into an m-dimensional descriptor $f_{geometry}$, such that objects with similar geometry are nearby in feature space and vice versa.

$$embed_{geometry} : \mathbb{Z}^{n\times 3} \mapsto \mathbb{R}^m,\ embed_{geometry}(n) = {}^n f_{geometry} \tag{34}$$

We denote the function that embeds a set of voxels into a feature vector as in equation 34. We implement this function using two different approaches, which we discuss below.

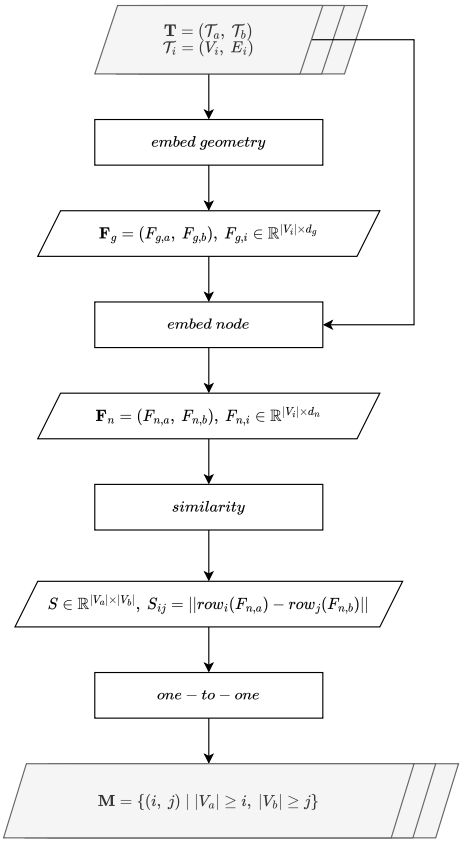

Figure 19: Diagram showing map matching methodology.

4.3.3 Spectral Features

Our first approach to geometric feature embedding uses spectral shape analysis. This approach uses the first n sorted, non-zero eigenvalues of the graph Laplacian, in our case of the neighbourhood graph of a room's geometry, as a geometric descriptor. To compute this we first convert each room's neighbourhood graph $\mathcal{G}$ to an adjacency matrix A and a degree matrix D. We then find the Laplacian matrix of the neighbourhood graph by subtracting its adjacency matrix from its degree matrix, as shown in equation 35.

$$L = D - A \tag{35}$$

After computing the Laplacian matrix we find its eigenvalues, sort them in ascending order and use the first 256 non-zero values as the descriptor.

4.3.4 Deep Learning

Our second approach to geometric feature embedding uses deep learning. Specifically, we use the LPDNet neural network architecture. This architecture is used for place recognition, it does so by learning descriptors, typically 2048 or 4096-dimensional, of point clouds that are theoretically independent of transformation, perspective and completeness. It does so by computing a local descriptor for every point in the point cloud and aggregrating them into a global descriptor. The LPDNet model we use is trained on outdoor maps which have different characteristics from indoor maps. However, the authors of LPDNet claim that a model trained on outdoor data can also effectively be used for indoor data. Figure 20 shows the network architecture of LPDNet.

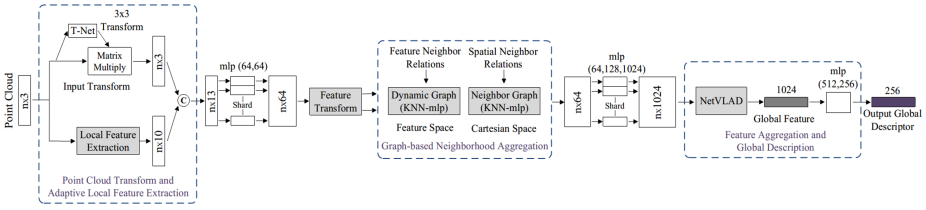

Figure 20: Diagram showing LPDNet network architecture.

4.3.5 Contextual Embedding

After computing a descriptor for each individual room we augment them by taking into account the descriptor of the neighbourhood. For every room we find their neighbours and merge their geometry into one voxel grid, for which we compute a new descriptor. We do this step multiple times for neighbours that are at most one or multiple steps away from the room. We then append the descriptors of the neighbourhood to the room's descriptor. By doing so we can distinguish between rooms with similar geometry but dissimilar neighbourhoods, which are often present in indoor environments.

4.3.6 Initial Matching

The above steps are applied to both partial maps. This gives us two sets of descriptors $\mathcal{E}_A$, $\mathcal{E}_B$ representing the embedding of the geometry of the rooms of both topometric maps.

To identify the most likely overlapping rooms between the partial maps we find the one-to-one mapping between the elements of $\mathcal{E}_A$ and $\mathcal{E}_B$ that maximizes the similarity (or minimizes the distances) between the chosen pairs. This is an example of the unbalanced assignment problem, which consists of finding a matching in a weighted bipartite graph that minimizes the sum of its edge weights. It is unbalanced because there may be more nodes in one part of the bipartite graph than the other, which means it is not possible to assign every node in one part to a node in the other.

To construct the weighted bipartite graph we first find the Cartesian product of the feature vectors

$$\mathcal{E}_{AB} = \mathcal{E}_A \times \mathcal{E}_B = \{(a, b) \mid a \in \mathcal{E}_A,\ b \in \mathcal{E}_B\} \tag{36}$$

We then compute the Euclidean distance in feature space between every pair of nodes in $\mathcal{E}_{AB}$, creating the cost matrix that represents the weighted bipartite graph

$$\mathbf{C} \in \mathbb{R}^{|V_a| \times |V_b|} \tag{37}$$

$$\mathbf{C}_{ij} = ||a - b||,\ (a,\ b) \in \mathcal{E}_{AB},\ a = \mathcal{E}_{A,\ i},\ b = \mathcal{E}_{B,\ j},\ \mathbf{C}_{ij} \in \mathbb{R}^{+} \tag{38}$$

We can then find unbalanced assignment using the Jonker-Volgenant algorithm. We denote the resulting matching between the nodes of both partial maps and their distance in feature space as a set of triples

$$\mathbf{M} = \{(i,\ j,\ d) \mid |V_a| \geq i,\ |V_b| \geq j,\ d = \mathbf{C}_{ij}\} \tag{39}$$

4.3.7 Hypothesis growing

In practice it is unlikely that every match in $\mathbf{M}$ is correct. However, we can use them as seeds to generate hypotheses similar to the approach described in Huang (Huang & Beevers, 2005). Starting at each initial match we get the neighbourhood of both nodes. We then construct a new cost matrix from the Euclidean distance between the embeddings of both neighbourhoods, again creating a weighted bipartite graph for which we can solve the assignment problem. By doing this we identify which neighbours of the nodes in the match are most likely to also match. We recursively apply this step to the matching neighbours to grow our initial matches into hypotheses. To decrease the risk of incorrectly identifying neighbourhood matches we constrain hypothesis growing in two ways. First, the cost of two potential matches must be below a given threshold c_{max}. Second, a newly identified match may not bring the existing matching too much out of alignment. To check this, we perform a registration (see next section) between the centroids of the geometry of the identified matches at every step of the hypothesis growing. If the error increases between steps, and the increase is too large such that $\triangle e \geq \triangle e_{max}$, then the matching is rejected. By adjusting the values of c_{max} and $\triangle e_{max}$ more or less uncertainty is allowed when growing hypotheses.

4.4 Map Fusion

4.4.1 Overview

The final step of the map merging process is map fusion, in general this means the problem of combining multiple partial maps into a global map. In our case, it specifically refers to the fusion of two partial topometric maps at the geometric and topological level to produce a global topometric map.

To achieve geometric fusion we designate one partial map as the source and the other as the target and find a transformation that aligns the two maps. We constrain the transformation to a rotation around the y-axis and a translation because most 3D scans of indoor environments are gravity-aligned, which means that it is not necessary to consider rotations around the x- or z-axis.

To find the transformation between the partial maps we first find the transformation between each pair of matched rooms. We do so by using RANSAC to find a global alignment which we then refine using the iterative closest point algorithm. Afterwards, we cluster the transformations and find the mean transformation of each cluster. We use the mean transformation which leads to the smallest difference between partial maps as our final rigid transformation. We then apply this transformation to the source map and fuse the geometry of the two maps into a global voxel grid map. Finally, we extract a new, global topometric map from the fused geometry. Figure 21 shows an overview of our map fusion approach. In the rest of this section we will describe our map fusion approach in detail.

4.4.2 Registration

The goal of registration is to find a rigid transformation τ between two point clouds that minimizes the error, as defined by an error function e, between them. For the error function we use point-to-plane distance, as shown in equation 41. In the context of indoor mapping data is usually aligned to gravity, with the direction of gravity being equal to the direction of the negative y-axis. We take this into account by constraining the transformation between partial maps to a translation along all three axes and a rotation around the y-axis. The rigid transformation can thus be expressed as a 4-dimensional vector, as shown in equation 40. Reducing the degrees of freedom of the problem from 6 to 4 can improve the alignment.

$$\tau = \underset{\tau}{argmin}\ e(\mathcal{P}, \mathcal{Q}) = \begin{bmatrix} \gamma \\ \mathbf{t} \end{bmatrix} = \begin{bmatrix} \gamma \\ t_x \\ t_y \\ t_z \end{bmatrix} \tag{40}$$

$$e = \sum_{k=1}^{K} ||((R(\gamma)\mathbf{p_k} + \mathbf{t}) - \mathbf{q_k}) \cdot \mathbf{n_k}||,\ \mathbf{p_k} \in \mathcal{P},\ \mathbf{q_k} \in \mathcal{Q} \tag{41}$$

Kubelka et al. (2022) gives a method for restating gravity-constrained alignment between two sets of points as a system of equations, which is shown in

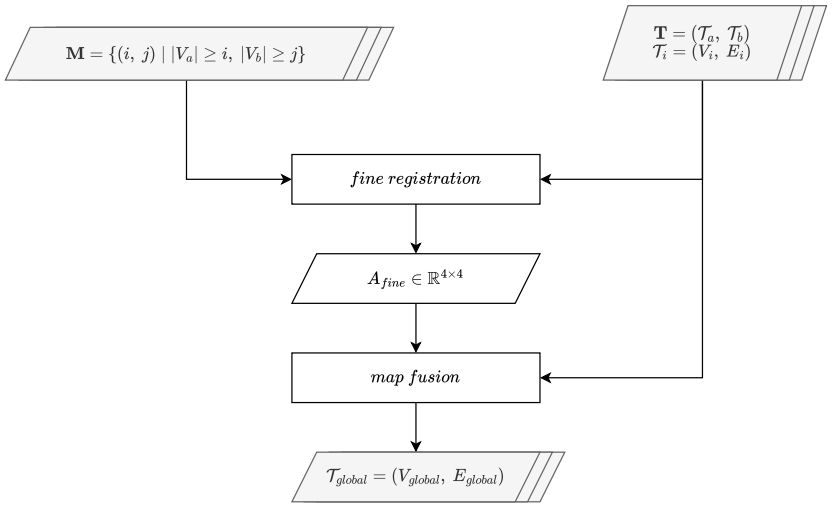

Figure 21: Diagram showing overview of map fusion methodology.

equations 42 and 43 (adapted to use y-axis instead of z-axis as the gravity direction). We can then solve this system of equations using least squares adjustment to find the optimal transformation τ.

$$c_k = (\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix} \mathbf{p_k}) \cdot \mathbf{n_k} \tag{42}$$

$$\sum_{k=1}^{K} \begin{bmatrix} c_k \\ n_k \end{bmatrix} \begin{bmatrix} c_k & n_k \end{bmatrix} \tau = \sum_{k=1}^{K} \begin{bmatrix} c_k \\ n_k \end{bmatrix} (\mathbf{d_k} \cdot \mathbf{n_k}) \rightarrow \mathbf{A}\tau = \mathbf{b} \tag{43}$$

$$transform : (\mathbb{R}^{m\times3},\ \mathbb{R}^{m\times3}) \rightarrow \mathbb{R}^4,\ transform(\mathcal{P},\ \mathcal{Q}) = \tau \tag{44}$$

The above is able to align two point clouds optimally if the correspondence between points is known exactly, meaning that the k-th point of both $\mathcal{P}$ and $\mathcal{Q}$ correspond to the same point in the real world. This is, however, usually not the case in real world scenarios. This means that registration requires another two steps, global and local registration, which we will discuss below.

Global registration The purpose of global registration is to find a rough alignment between point clouds, which can then be further refined in the local registration step. To do so we use a RANSAC based approach. This algorithm works as follows. For every point in the point clouds $\mathcal{P}$ and $\mathcal{Q}$ compute a feature embedding that can be used to find similar points in the other point clouds. We use Fast Point Feature Histograms (FPFH) features, which are commonly used for this purpose. We then randomly select 3 points from $\mathcal{P}$ and find the corresponding points in $\mathcal{Q}$ which have the smallest distance in

feature space. We then compare the triangles formed by both sets of points. If the edge lengths of both triangles are too dissimilar, meaning that the ratio of their lengths is outside of a predetermined range, then the selected points are discarded and new ones are selected. If not, we find the gravity-constrained rigid transformation between the 3 pairs. We then evaluate how well the rigid transformation aligns the two point clouds by finding the mean distance of every point in $\mathcal{P}$ to its nearest neighbour in $\mathcal{Q}$. If the mean distance is smaller than the previous smallest mean distance then we store the transformation. We repeat this for a set amount of iterations or until a mean distance threshold is reached. The rigid transformation with the smallest mean distance is then used for the next step, local registration.

Local registration The purpose of local registration is to refine the alignment found in the global registration step. We use the iterative closest point (ICP) algorithm to achieve this. The ICP algorithm is widely used, a detailed description of how it works can be found in (CITATION HERE). The only major modification is that we use the gravity-aligned transformation with the point-to-plane error function described above at each iteration.

Transform concatenation After finding the global and the local transformations we can find the final transformation τ by multiplying their by 4 transformation matrices. We denote the function that converts τ to a 4 by 4 transformation matrix as in equation 45. Equation 46 shows how to compute the final transformation matrix, note that the order of multiplication is important, as matrix multiplication is not commutative.

$$T : \mathbb{R}^4 \to \mathbb{R}^{4\ times4} \tag{45}$$

$$T(\tau) = T(\tau_{local})T(\tau_{global}),\ T : \mathbb{R}^4 \to \mathbb{R}^{4\times4} \tag{46}$$

4.4.3 Geometric fusion

We apply the above steps to the geometry of each room and its match if their distance in feature space is below a threshold. This gives us a 4-dimensional vector representing the transformation between matches. We discard the matches where the registration error is too high and cluster the remaining transformations using the DBSCAN algorithm. This gives us multiple clusters of transformations that, within a cluster, result in a similar alignment between partial maps. For each of these clusters we find the mean transformation, apply it to the partial map and compute the point-to-plane error between the partial maps. We then select the cluster with the lowest error and use its mean transformation to align the geometry of the partial topometric maps.

4.4.4 Topological fusion

After the geometric fusion step the geometry of the topometric maps is brought into alignment but their graphs haven't been fused yet. Fusing the graphs directly using the identified matches is possible but does not guarantuee a good global topometric map. This is because the geometric fusion may have changed the partial maps in a way that the global geometric map's topology is greater than the sum of the partial maps' topology. As a result, we have to reextract the topology from the results of the geometric fusion. To do so, we reuse the navigation graphs of the partial maps as the navigation graph of the global map. We also reuse the optimal viewpoints from both partial maps. This means that only the room segmentation step needs to be redone. The result is a global topometric map created from the fusion of two partial topometric maps.

5 Results

We evaluated the algorithm described in the methodology section on several datasets. In this section we describe these datasets and show the results we achieved with them. The shown results are divided into three sections based on the major components of our methodology: map extraction, map matching and map fusion.

5.1 Datasets

In this section we describe the datasets that we used to test our algorithm. We also describe how we prepared the ground truth datasets so as to be able to compare our results to them and objectively measure their performance.

5.1.1 Simulated Scan

To objectively evaluate the performance of our methodology it is necessary to compare the results to a ground truth. This ground truth must contain the following elements: room labels and their topological relationships to evaluate map extraction, room matches between partial maps to evaluate map matching, and the transformations between partial maps to evaluate map fusion. When capturing real-world data, the second and third elements are especially difficult to determine. Furthermore, little pre-existing data is available that contains exactly these elements. To solve this problem we simulate partial maps from annotated global maps from various sources. The annotations are integer labels for every point representing the ground truth room segmentation and a graph representing the ground truth topological map.

We create the partials maps by manually defining a trajectory for each desired partial map and simulating an agent moving along them, scanning the global map at a set interval. We do this by using the DDA algorithm described in the methodology section. This allows us to objectively evaluate our approach for a large number of different scenarios at the cost of missing some of the subtleties inherent in non-simulated partial maps, such as changes in the environment and measurement error. To mitigate this, we apply pre-processing steps to the global map. These pre-processing steps are: random removal of points, adding noise to points and random rotation and translation of the point cloud. The latter's purpose is to simulate the unknown transformation between real-world partial maps. By comparing the results of our approach to the annotations in the ground truth global map we can objectively measure the performance of map extraction, matching and fusion. Figure 22 shows an example global map with simulated viewpoints. Figure 23 and 24 show two simulated partial maps created from a single map.

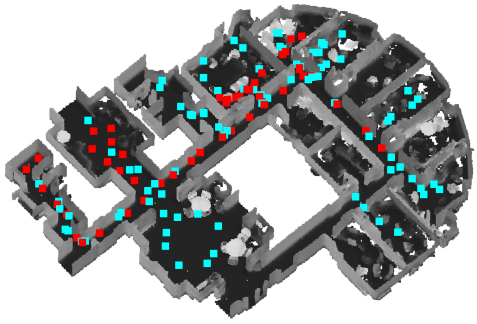

Figure 22: Global map with simulated viewpoints. Each coloured dot represents a viewpoint, the union of all views with the same color forms a partial map.

5.1.2 Stanford 3D Indoor Scene Dataset

The Stanford 3D Indoor Scene Dataset (S3DIS) consists of a collection of 3D scans of six different indoor environments (Area 1 through 6) and the raw measurements used to create them. The environments all span a single floor. Each environment scan in its original state consists of a number of point clouds, one for each room in the building. We merge these separate point clouds into a single cloud but retain the room labels as point attributes, as we use these as our ground truth room labels for the map extraction step. We manually created the topological graph of each area. The S3DIS was captured using high-end 3D scanners and thus has a high point density.

5.1.3 Collaborative SLAM Dataset

The Collaborative SLAM Dataset (CSLAMD) is a dataset meant specifically for collaborative SLAM. It consists of three environments (House, flat and lab), each consisting of multiple partial maps and their ground truth transformations. Two of the environments consist of multiple storeys. The partial maps were captured using a low-end 3D scanner and thus has a low point density. We merge the partial maps of each environment into a single global map to create the simulated partial maps described above. We then manually annotate each environment with our interpretation of an appropriate room segmentation and manually create the topological graph.

5.1.4 Various Sources

Finally, we also include datasets from various sources. These include a two-storey house with basement and an office conference room, both captured using high-end 3D scanners. We annotated these datasets by hand according to our interpretation of the spaces and their segmentation into rooms and manually create the topological graph.

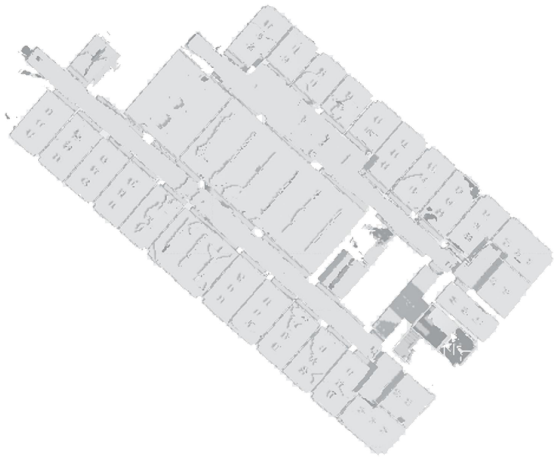

Figure 23: First simulated partial map extracted from S3DIS area 1 dataset.

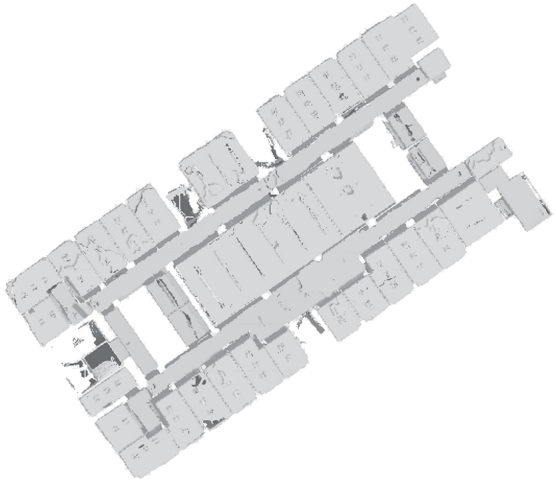

Figure 24: Second simulated partial map extracted from S3DIS area 1 dataset.

5.2 Map Extraction

In this section we show the results of our map extraction approach. To evaluate the results of our map extraction approach we compare the extracted topometric partial maps to the ground truth global map. For each node's geometry in a topometric map we find its Jaccard index with every node in the global map's geometry. This gives us a weighted bipartite graph, one part being the nodes in the partial map and the other being the nodes in the global map, with the weight representing the Jaccard index between nodes' geometry. We then assign each

node in the partial map to a single node in the global map such that the sum of weights is maximized using linear assignment as described in the map matching section. This gives us the correspondence between nodes in the partial map and nodes in the global map. Afterwards, we compute the mean Jaccard index of the correspondences. This metric is called Mean Intersection over Union (MIoU).

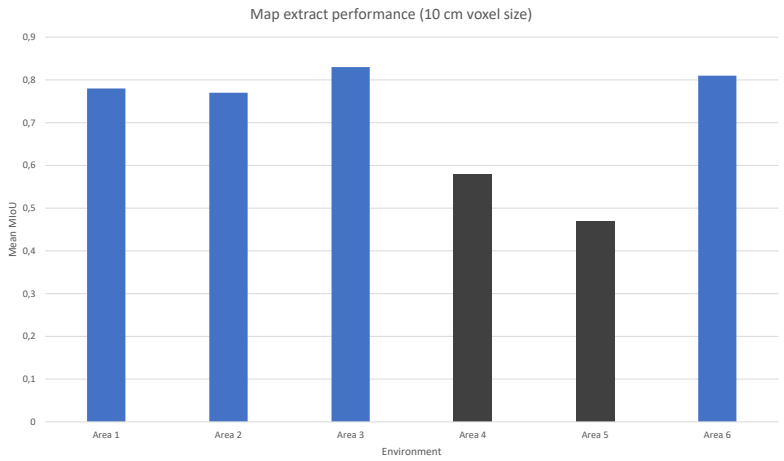

Figure 25: First topometric map extracted from S3DIS area 1 dataset.

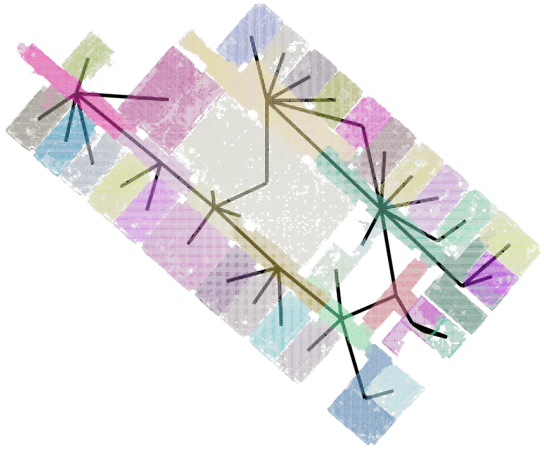

Figure 26: First topometric map extracted from S3DIS area 1 dataset.

5.3 Map Matching

In this section we show the results of our map matching approach. Using to partial map to global map node correspondences described in the previous section we can distinguish incorrect matches from correct matches; a match between two partial maps is correct if both nodes in the match correspond to the same node in the global map. With this information we evaluate our map matching approach based on four metrics: precision, accuracy, recall and F1.

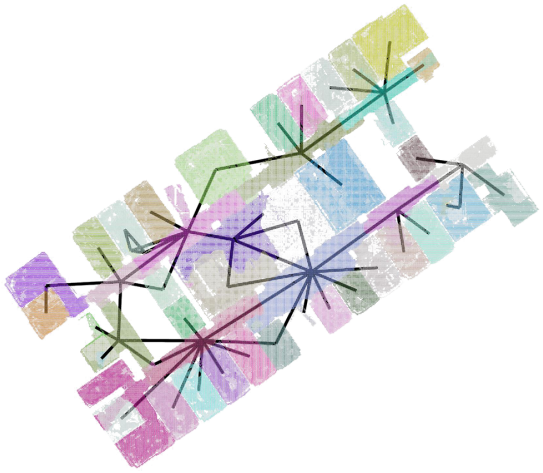

Figure 27: Second topometric map extracted from S3DIS area 1 dataset.

5.4 Map Fusion

In this section we show the results of our map merging approach. We evaluate the performance of our map merging approach by comparing the computed transformation to the transformations applied to the ground truth when simulating the partial maps.

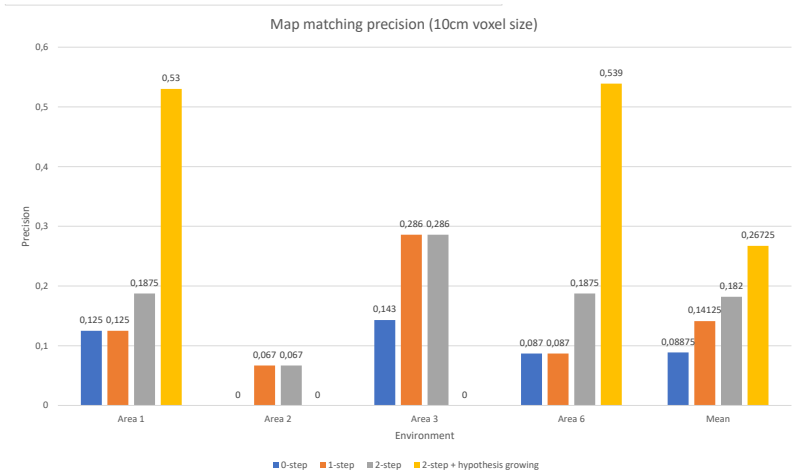

Figure 28: First topometric map extracted from S3DIS area 1 dataset.

Figure 29: First topometric map extracted from S3DIS area 1 dataset.

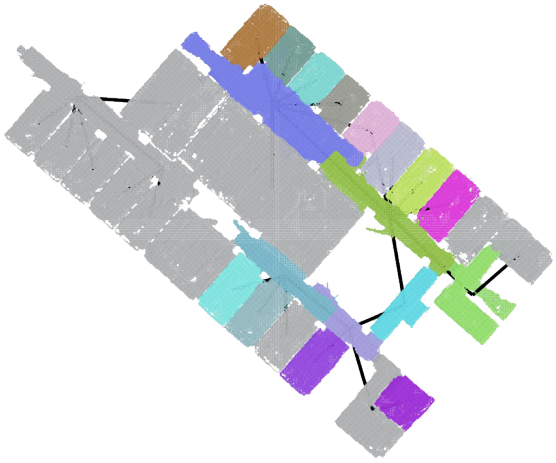

Figure 30: Second topometric map extracted from S3DIS area 1 dataset.

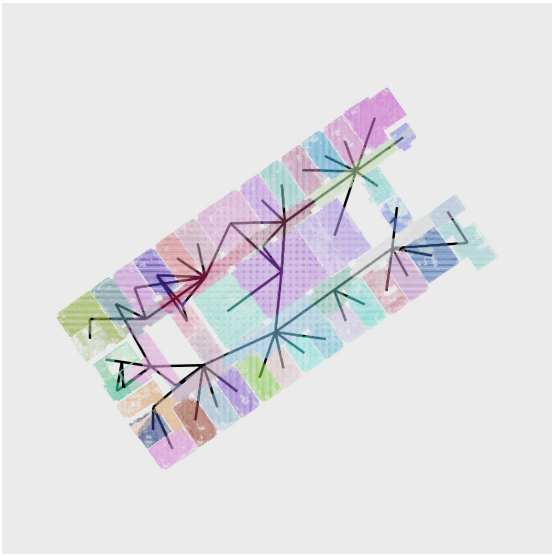

Figure 31: Second topometric map extracted from S3DIS area 1 dataset.

6 Discussion

6.1 Map Extraction

In the previous section we showed the results of our map extraction approach. In this section we will discuss these results. For the majority of tested environments the resultant room segmentation closely matches the ground truth room segmentation. This is especially the case in environments where rooms have clear delineations; environments where rooms have walls between them and are only connected by small openings. The results of our room segmentation approach match less closely in environments where this is not the case. Our approach often splits single rooms that are large in one or multiple dimensions, such as hallways or auditoriums, into multiple rooms. Additionally, rooms where there are obstructions to the view from inside the room are also split into multiple parts. However, these effects do not necessarily indicate a failure of our approach. The segmentation in the ground truth data is based on human intuition about what separates a room from its neighbours. Although room segmentation based on visibility clustering often closely matches this intuition it is inherently different as it does not take into account the intended use of rooms. Where humans might recognize that a long hallway or a large hall serves a single purpose, and should therefore be considered as the same room, visibility clustering fails to take this subjective interpretation of purpose into account. Nevertheless, the objective visibility clustering approach comes remarkably close to the subjective human approach. The opposite also occurs, where two or more rooms that are separate in the ground truth data are not separate in the room segmentation. This mostly occurs when there are no obstructions between two adjacent rooms. This can often be resolved by changing the clustering parameters. However, when the only separation between two rooms is based on purpose and not on visibility then our approach cannot separate them.

A common failure mode of our approach, which causes room segmentation to fail completely, is when the input point cloud data is of insufficient density to construct a connected navigation graph. In this case, the voxels belonging to the navigation graph are identified correctly but there are gaps between voxels. This can be solved by increasing the size of the kernel used for constructing the neighbourhood graph of the navigable voxels. However, this has the side effect that voxels that are not actually navigable are added to the navigation graph. The result is that low elevated surfaces with sloping sides, beds for example, are added to the navigation graph. While this usually does not have a large effect on map extraction in extreme cases it can also cause the ceiling to become part of the navigation graph. This will usually cause significant errors in room segmentation, as the view from above the ceiling towards the rest of the map is often completely unobstructed.

Another way that room segmentation may fail is when stairs have very shallow treads and steep rises (respectively the horizontal and vertical part of its steps). This causes the stick kernel approach to fail to label the stairs' voxels as navigable, which means it will not be included in the navigation graph. This is

because the wide part of the stick kernel placed on one step may intersect with the next step. If there are no other connections between two storeys then this will cause one or multiple storeys to become disconnected from the navigation graph, excluding it from the extracted topometric map. This problem can be resolved by changing the dimensions of the stick kernel. However, this may in turn cause other problems. Increasing the height of the thin part of the stick kernel causes low elevated surfaces with sloping sides to be included in the navigation graph, as described in the previous paragraph. Decreasing the radius of the stick kernel's wide part will include parts of the map that are not actually navigable in the navigation graph. The problem can also be solved by increasing the size of the kernel used to construct the navigable voxels' neighbourhood graph to force the stairs' voxels to become connected even though some are missing but this causes the same issues as described in the previous paragraph.

Differences between the ground truth topological graph and the extracted topological graph are caused by differences in room segmentation. One such case is when a hallway connected to a room is split into multiple rooms around the opening towards the connected room. This will result in a triangular subgraph between the two parts of the hallway and the connected room, which in reality should just be a single edge between the hallway and the room.

6.2 Map Matching

In this section we will discuss the results of our map matching approach. We will discuss this in two parts. The first concerns the results for feature embedding, the second concerns the hypothesis growing step.

Feature embedding As seen in the results section the performance of initial matching strongly depends on the feature embedding approach. We find that the deep learning approach gives the best results here as it is able to handle large differences in segmentation (cases where the voxels assigned to the same room between two maps have a large overlap but do not match exactly) and completeness (cases where a room in one partial map has not been captured completely). In contrast, the engineered feature and the spectral approaches are not able to deal with incompleteness and to a lesser degree differences in segmentation. However, even for the deep learning approach incompleteness and segmentation differences have a significant negative impact on performance. In some cases this problem is resolved by the graph convolution step due to the added information about the rooms' neighbours. However, the results of this are inconsistent and it is currently often better to not apply graph convolution at all.

Another factor that a significant negative impact on feature embedding performance is the similarity of rooms. In cases where there are many near identical rooms, which is often the case in environments like offices and hospitals feature embedding fails to distinguish between them. This is especially the case when a large voxel size is used because it removes details like furnishing and clutter that may help distinguish between rooms. When this is the case only the shape

of the room can be used to identify it which may match very closely to similar rooms. When similar rooms are not adjacent graph convolution can reduce this problem, especially when the adjacent rooms are very distinctive. However, the opposite is true when similar rooms are adjacent. In this case, graph convolution makes the rooms' already similar feature embedding even more similar, making it even harder to distinguish between them. This poses a problem because graph convolution can't be applied selectively and it is currently unknown how to predict when it will improve performance and when it won't. Based on our observations, all three embedding approaches suffer with distinguishing similar rooms, graph convolution or not, with deep learning performing slightly better than the other two.

Hypothesis growing Based on our results we find that hypothesis growing has the potential to significantly improve map matching performance. However, its performance greatly depends on the quality of the feature embedding. If the initial matching is completely incorrect then hypothesis growing also fails. Even if some initial matches are correct, the performance of hypothesis growing still depends on the quality of the feature embedding. An exception to this is when the initial match used to grow a hypothesis is at the end of a linear chain of rooms. In this case the growing step will succesfully match all rooms in the chain given that the feature embedding between two matches does not fall under the similarity threshold.

We also find that the transformation estimation step is able to constrain region growing to give more reasonable results by preventing matches from being made that would bring the existing matching out of alignment. Adjusting the transformation difference threshold upwards allows the region growing to grow further while increasing the risk that an incorrect match is made. In reverse, adjusting it downwards makes region growing more restrained and decreases the risk of incorrect matches. In the ideal case with no differences between partial maps and their segmentation the threshold could be set to zero as any correct match would align perfectly with the existing matches. However, incompleteness of data and error between partial maps causes the centroids of two matching rooms to be in different positions, introducing error into the alignment. From this it follows that incompleteness, and to a lesser degree error, also has a significant impact on the hypothesis growing.

6.3 Map Fusion

6.4 Future Works

In this section we discuss our recommendations for future developments of the three major components of this thesis: map extraction, map matching and map fusion. We make these recommendations based on the achieved results and our research during the creation of this thesis.

6.4.1 Map extraction

Room segmentation As mentioned before, the current room segmentation approach differs from how humans identify rooms as it does not take into account the perceived purpose of the room. Taking both visibility and purpose into account could lead to a segmentation that more closely matches one a human would perform, but more importantly, one that is more consistent between partial maps and more robust to incompleteness and error. Assuming that a computer can succesfully infer a room's purpose based on its voxelized representation, large rooms such as hallways that are now arbitrarily divided based on clustering could be merged into one whole. Inferring a room's purpose is a subjective task that would be very hard to solve using traditional techniques. To achieve this, we suggest training a deep learning model that is suitable for segmentation of voxel or point cloud representations, such as PVCNN or DGCNN, on a manually labeled ground truth dataset.

Robust topological graph extraction One of the major bottlenecks of our approach is the extraction of the topological graph. If this step fails then map extraction fails and map matching becomes impossible. Thus, in the future it would be important to identify an approach to topological graph extraction that is robust to the failure modes described in section 6.1. This would include a way to interpolate the navigation graph to fill in any missing holes that cause it to become disconnected. This could be done using interpolation techniques from image processing applied to 3-dimensional data or more advanced techniques such as the PCN network discussed in section **??**. In both cases the main challenge is differentiating between voxels that should be present but are missing and voxels that should not be; making the wrong choice could lead to worse results than no interpolation at all. For example, a small gap in the floor can be either a piece of missing data or a gap between walls. Solving this challenge could drastically improve the robustness of our map extraction approach and should be considered in the future.

Hierarchical topological representation Our current approach to map extraction results in a topometric map with a 'flat' graph representing the environment's topology. In reality, indoor environments can be considered as complex multi-level hierarchies. For example, a building can be divided into storeys which contain rooms which contain areas. As such, the structure of an indoor environment can also be represented by a hierarchical graph. The extra information contained in such a graph could be applied to improve feature embedding performance. For example, two rooms are more similar if their storeys are also similar than if they are not. Various techniques have been proposed in the literature surrounding this subject but none so far are based on visibility clustering. We hypothesize that by applying hierarchical clustering to visibility it is possible to extract a hierarchical structure of the environment. Whether this is true and what the characteristics of the resultant map are could be a valuable topic of research. A hierarchical topological representation could allow

or require different methods for hypothesis growing and map fusion. For the former, work in the area of hierarchical graph matching could be applied. The latter is, to the knowledge of the author, still unresearched.

Volumetric representation Our current approach only represents the surface of the geometry of the environment. This is because 3D scanners only capture that part of the environment. In reality, indoor environments are enclosed volumes. A possible improvement to our approach would be to describe the environment's geometry volumetrically, where each occupied voxel represents a volume within the building that is not obstructed. This would require a method to extract the volume from the surface geometry. Various research into this topic exists (REFERENCES HERE) but they fail when parts of the ceiling or floor are missing from the map, which is often the case. They also make assumptions such as constant storey height and only horizontal floors, which are often not the case in reality. Using a volumetric representation of the environment has a number of benefits. Navigation would no longer only be possible on the floor but throughout the entire volume. In reality most scanners use the floor to navigate but the advent of drones that operate indoors might change this. In addition, a volumetric representation might improve feature embedding performance as it describes the room more completely (DOES THIS MAKE SENSE?). Another avenue of research that moving to a volumetric approach would require would concern efficiently storing and processing the exponentially larger amount of data used in doing so. While this subject has been considered in this research by using sparse voxel octrees, variations on this or other data structures might be more effective. For example, implementations of sparse voxel octrees for GPUs exist.

6.4.2 Map Matching

6.4.3 Map Fusion

6.5 Conclusion

In our research, we tried to answer how the properties of 3D topometric maps of indoor environments can be applied to the map merging problem. In this section we will give our conclusion to each of our subquestions that together will answer our main question.

Our first subquestion asked how to extract these topometric maps from point clouds. We conclude that a visibility clustering approach to segment the map, combined with voxel convolution using specialized kernels to extract the map's topology, can effectively be used to extract topometric maps. However, this approach is currently very sensitive to map quality which opens up new avenues of research.

Our second subquestion asked how to find matches between partial topometric maps to identify their overlapping areas. After comparing various descriptor approaches we conclude that the deep learning approach approach gives the best results. We also find that embedding the context of a room into the room's descriptor improves matching performance. So does hypothesis growing in the average case, but it may also cause map matching to fail completely. Based on the above we further conclude that the topological aspect of topometric maps can be used to increase map matching performance.

Our third subquestion asked how to fuse the partial topometric maps after matches have been identified. We tried two registration approaches, ICP and DCP and found that ICP is better suited for its ability to constrain the transform without requiring retraining. We conclude that using ICP to register the room submaps can also be used to reject incorrect matchings.

With the above conclusions we can answer our main research question: partial topometric maps can be extracted from partial point clouds by using visibility clustering and voxel convolution, matches can are best found using a combination of deep learning descriptors, context embedding and hypothesis growing, using these matches the topometric map submaps can then be individually fused using ICP registration to create the global map and reject incorrect matches.

References

Huang, W. H., & Beevers, K. R. (2005, August). Topological Map Merging. *The International Journal of Robotics Research*, *24*(8), 601–613. Retrieved 2021-11-29, from http://journals.sagepub.com/doi/10.1177/0278364905056348 doi: 10.1177/0278364905056348

Kubelka, V., Vaidis, M., & Pomerleau, F. (2022, March). *Gravity-constrained point cloud registration.* arXiv. Retrieved 2022-08-29, from http://arxiv.org/abs/2203.13799 (arXiv:2203.13799 [cs])

Kuipers, B. (1978, April). Modeling spatial knowledge. *Cognitive Science*, *2*(2), 129–153. Retrieved 2022-01-10, from https://www.sciencedirect.com/science/article/pii/S0364021378800032 doi: 10.1016/S0364-0213(78)80003-2

Kuipers, B., & Byun, Y.-T. (1988). A Robust, Qualitative Method for Robot Spatial Learning.

Thrun, S. (1998, February). Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, *99*(1), 21–71. Retrieved 2022-01-10, from https://www.sciencedirect.com/science/article/pii/S0004370297000787 doi: 10.1016/S0004-3702(97)00078-7

Volodine, T. (2007). *Point Cloud Processing Using Linear Algebra and Graph Theory* (Unpublished doctoral dissertation).