Map Merging of Heterogeneous Topometric Maps Using Attributed Graph Embedding

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

1 Introduction

Collaborative mapping allows multiple agents to work together to create a single, global map of their environment. By working together large areas can be mapped in a short amount of time. Most existing research has focused on when mapping agents are homogeneous, meaning they sense their environment and behave similarly (Andersone, 2019). However, there are situations where mapping with heterogeneous agents can be advantageous (Hermann et al., 2016). For example, a human carrying lightweight, low-end sensors collaborating with a robot carrying heavy, high-end sensors to map an environment where some areas are not accessible by humans. By compensating for the weaknesses of one agent with the strengths of another more environments and situations can be handled.

Existing collaborative mapping approaches are not well suited for mapping with heterogeneous agents as they often rely on agents being able to communicate with eachother. This is especially the case in indoor environments where external positioning signals are highly attenuated. In this case, a global map can only be created by merging the partial maps of the environment created by each agent individually based on their overlapping areas. This is called map merging (see figure 1). When partial maps are created by heterogeneous agents they might represent different aspects of the environment at different a different scale, resolution or accuracy, which further complicates map merging as overlapping areas may not appear the same between partial maps.

Figure 1: Partial maps (1) captured by three different agents and resultant global map (2) after map merging.

In this thesis we propose to use both the topological relationships of indoor environments, meaning the connectivity between distinctive places, and their metric characteristics, the geometry of the environment, to solve the heterogeneous map merging problem. We do this by extracting hybrid 3D topometric maps from heterogeneous partial maps. We then use both the topological and metric characteristics of the partial maps to robustly identify overlapping areas that might appear differently due to being captured by heterogeneous agents. Our hypothesis is that the connectivity of places within the environment are identifiable between heterogeneous 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 topological approach, despite geometrical differences between heterogeneous partial maps. The most important contributions of our work are: 1) applying 3D topometric map merging to indoor environments. 2) extracting 3D topometric maps from heterogeneous partial maps.

2 Research Questions

2.1 Main question

To what extent can topometric representations of indoor environments be applied to heterogeneous map merging?

2.2 Subquestions

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

2.3 Scope

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

Previous research on mapping and map merging has used various map representations are used depending on the map's intended purpose (Tomatis et al., 2003; Huang & Beevers, 2005; Bonanni et al., 2017; Gholami Shahbandi & Magnusson, 2019). According to Andersone (2019) and Yu et al. (2020) these representations can be subdivided into three types: metric-, feature-, and topological maps. Hybrid maps that are combinations of two or more map types also exist, such as topological-metric maps (Yu et al., 2020). Map types that are not one of the three main types or a hybrid are rarely used (Yu et al., 2020). In this section we will discuss the characteristics of the metric, topological and topological-metric map representations and the work that has been done on extracting and merging them.

3.1 Metric Maps

Metric map merging is a mature area of research and thus various approaches have been proposed, many of which use a variation of the Iterative Closest Point (ICP) algorithm. This algorithm finds the transformation between partial maps by iteratively applying rigid transformations that minimize the distance between point-pairs (Rusinkiewicz & Levoy, 2001). Since its first introduction a number of variations on the ICP algorithm have been proposed that improve its accuracy and performance. An example of this is the NICP algorithm, which improves data-association by taking into account the normal vector of a point's neighbourhood (Serafin & Grisetti, 2015).

Another group of approaches to metric map merging use feature matching to identify overlaps between maps. These approaches try to extract distinctive features in the metric map, such as corners, lines, planes or other points of interest, e.g. SIFT, SURF or Harris points (Andersone, 2019). Feature matching approaches have the advantage over ICP-based approaches with regards to required computational power (Andersone, 2019). Some feature matching approaches are better suited for different environments and input data. For example, the feature extractor approach by Li & Olson (2010) is scale-independent and the approach of Yang et al. (2016) is able to deal with differences in resolution. Yang et al. (2016) also proposes a combination of feature-based and ICP-based metric map merging that uses features to find a rough alignment which is then further refined using a variation on the ICP algorithm, as shown in figure 2.

In conclusion, much work has been done on the merging of metric maps using brute-force ICP-based methods. However, these methods have several downsides and are thus losing popularity in favour of feature-based approaches. Research on many different kinds of features is available, with state-of-the-art approaches being able to handle large differences in appearance of features. Both approaches can also be combined to compensate for their respective downsides.

Figure 2: Illustration of combined feature-based and ICP metric map merging approach from Yang et al. (2016).

3.2 Topological(-Metric) Maps

Various approaches for extracting topological maps from raw sensor data or metric maps have been proposed. Kuipers & Byun (1991) proposes identifying distinctive places, vertices of the topological map, directly from sensor data by finding local maxima of a distinctiveness measure within a neighbourhood. Edges are identified by having the robot try to move between vertices, if this is possible an edge is created. Local metric information in the form of an occupancy grid is associated with the nearest vertex in the graph, resulting in a hybrid topological-metric map. Note that this approach is dependent on the mapping agent's exploration strategy. As heterogeneous agents can't be assumed to behave in a similar way, approaches based on exploration strategy are not directly applicable for the purposes of this thesis.

Thrun (1998) extracts a 2D topological map from a 2D metric map by identifying narrow passages with the use of a Voronoi diagram. They then partition the metric map into areas divided by passages, which are respectively the vertices and the edges of the graph. Again, metric information is associated to create a hybrid topological-metric map. This is the first approach that is independent of exploration strategy.

Ochmann et al. (2014) describes extracting hierarchical topological-metric maps directly from point clouds. In the case of Ochmann et al. (2014) the hierachy is divided into four encapsulating layers, building - storey - room - object. Entities within the graph are represented as vertices, with edges representing the topological and spatial relationships between entities. Each vertex is linked to a local metric map of the entity's geometry. All entities are also linked to the layer above them that they are in, e.g. objects are linked to their encompassing room, which is in turn linked to the storey it is in.

Gorte et al. (2019) provides a novel approach for extracting the walkable floor space from a 3D occupancy grid across multiple storeys. They do so by first applying a 3D convolution filter using a stick-shaped structuring element to extract the parts of the floor without obstructions. They then apply an upwards dilation to connect steps of stairways into a connected volume. Because the topology of the environment depends on the traversability between spaces, the extraction of navigable floor space is essential for the extraction of topological maps.

He et al. (2021) describes an approach for extracting a hierarchical topological-metric map with three layers: storey - region - volume, from an occupancy grid map. To extract the map they use a novel approach to room segmentation using raycasting. A downside of their methodology is that it depends on the presence of ceilings in the metric map, which are often not captured in practice when using handheld scanners. They also describe a novel approach for storey segmentation using a peak detector on the metric map's histogram of z-coordinates. The results of their approach are shown in figure **??**.

In comparison, relatively little work has been done on the subject of topological(-metric) map merging. Dudek et al. (1998) first proposes an approach to topological map merging which depends on a robot meeting strategy to merge partial maps created by each robot. When new distinctive places are recognized at the frontier of the global map the other robots will travel towards it and synchronize their maps. As with other early approaches to map extraction and map merging, this one also depends on a coordinated exploration strategy, making it unsuitable for the purposes of this thesis.

The work of Huang & Beevers (2005) is a significant milestone in topological map merging, demonstrating that topological maps can be merged using both map structure and map geometry. They first identify vertex matches by comparing the similarity of their attributes, such as their degree, and the spatial relationships of incident edges. Vertex matches are then expanded using a region growing approach, where every added edge and vertex is compared for similarity and rejected if too dissimilar. The results are multiple hypotheses for overlapping areas between partial maps. They then estimate a rigid transformation between the partial maps based on each hypothesis. Afterwards, hypotheses that result in similar transformations are grouped into hypothesis clusters. They then select the most appropriate hypothesis cluster by using a heuristic that includes the number of vertices in the cluster, the error between matched vertices after transformation and the number of hypotheses in the cluster.

Bonanni et al. (2017) provides a unique approach to topological-metric map merging using the pose graph of mapping agents as the topological component and the point cloud captured at each node as the metric component. Matches between nodes are identified by computing the similarity of their associated point cloud. In comparison to most other map merging approaches they fuse the maps using a non-rigid transformation, meaning the partial maps are deformed to improve their alignment. This gives their approach the ability to correct inconsistencies between partial maps that might be caused by differ-

ences between scanning agents.

To conclude, a variety of representations of topological and topological-metric maps have been proposed, as well as ways of extracting them from metric maps. Early approaches to topological and topological-metric map extraction depend on coordinated robot exploration strategies and only work in 2D, making them unsuitable for the purposes of this thesis. State-of-the art approaches are trending towards 3D hierarchical representations of buildings as technology improves and more ways of dealing with 3-dimensional data are discovered. However, little work has been done on extracting topological maps from heterogeneous metric maps, as well as on extracting them from incomplete partial maps. Relative to the amount of work on the extraction of topological(-metric) maps of indoor environments, less has been done on merging them. Most approaches compare labels of vertices and their incident edges to find matches. To our knowledge, Bonanni et al. (2017) is the only work that identifies vertex matching by comparing local metric map geometry. It is also the only approach that is able to handle deformed partial maps.

4 Methodology

4.1 Map Representations

4.1.1 Point Cloud

An unordered collection of points representing the geometry of an object or environment in 3D euclidean space, defined as $\mathcal{P} = \{p_i\}_{i=1}^{n}, p_i \in \mathbb{R}^3$, where n denotes the number of points (Volodine, 2007).

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 grid, a voxel grid, and its associated properties. For example, a voxel may contain information about whether it is occupied and what its color is. We consider a voxel as a three-dimensional vector representing its coordinates along the x, y and z axes of the voxel grid.

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 $e_l \in \mathbb{R}^+$. A voxel $\boldsymbol{v} = (x, y, z) \in \mathbb{Z}^{3+}$ represents a subvolume of V bounded by a single cell. The minimum and maximum bounds of this subvolume are given by $\boldsymbol{v_{min}} = V_{min} + \boldsymbol{v} * e_l$, and $\boldsymbol{v_{max}} = V_{min} + (\boldsymbol{v} + 1) * e_l$. The voxel's centroid is given by $\boldsymbol{v_c} = (\boldsymbol{v_{min}} + \boldsymbol{v_{max}}) * 0.5$. Given a point $\boldsymbol{p}$ within the bounds of V, the corresponding voxel is given by $\boldsymbol{v_p} = (\boldsymbol{p} - V_{min})//e_l$, where // denotes integer division. A property of a voxel is given by the function $\mathcal{V}_{property} : \mathbb{Z}^{3+} \mapsto \mathbb{R}^{m*n}$.

Due to the regularly spaced nature of the voxel grid a voxel coordinate only consists of integer values. Voxel $\boldsymbol{v_{V_{min}}} = (0, 0, 0)$ 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 $\boldsymbol{v_{V_{max}}} = (V_{max} - V_{min})//e_l$. We define a voxel grid as a set of occupied voxels $\mathcal{V} = \{v_i\}_{i=1}^{n},\ n \in [1, \prod \boldsymbol{v_{V_{max}}}]$ with an associated bounded volume $V_{\mathcal{V}}$ and edge length e_l. Figure 3 shows an example voxel grid and its components.

Sparse Voxel Octree Several operations on voxel grids benefit from using a spatial index, including range searching, 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 voxel octree, the leaf nodes represent individual voxels. In a

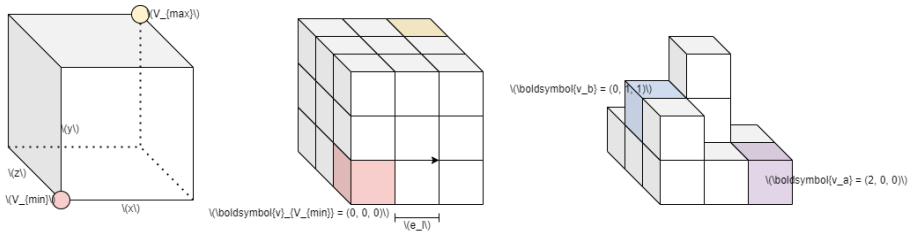

Figure 3: A voxel grid and its components.

sparse voxel octree only the octants which are occupied are represented in the tree.

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 coordinates into a single binary number, called a Morton code. The ordered vector of Morton codes gives the Morton order. We define the Morton order of a voxel grid as $M_{\mathcal{V}} = \{m_i\}_{i=1}^{|\mathcal{V}|},\ m_i < m_{i+1},\ m_i \in \mathbb{Z}^+$. We then divide the Morton order into buckets with width 8, such that each bucket contains at most 8 Morton codes, with a maximum difference of 8, in Morton order. 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, a sparse voxel octree is constructed.

We denote the function that returns all n voxels within range r of a voxel as $radius : \mathbb{Z}^{3+},\ \mathbb{R} \mapsto \mathbb{Z}^{n\times 3}$.

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 computing a function $\mathcal{K}_f : \mathcal{K}_w \mapsto \mathbb{R}^{m*n}$ over its neighbours. The neighbourhood can be a radius around the voxel, its Von Neumann neighbourhood, its Moore neighbourhood, or any other arbitrary shape. We can define a kernel $\mathcal{K}$ as a voxel grid, with an associated weight for each voxel $weight : \mathbb{Z}^{3+} \mapsto \mathbb{R},\ v \in \mathcal{K}$ and an origin voxel $\boldsymbol{o}_{\boldsymbol{\mathcal{K}}} \in \mathbb{Z}^{\not\Vdash}$. To apply a kernel to a voxel we first translate the kernel so that its origin lies on the voxel, such that $\mathcal{K}_{\boldsymbol{v}} = \{\boldsymbol{v}_{\boldsymbol{\mathcal{K}}} + (\boldsymbol{v} - \boldsymbol{o}_{\boldsymbol{\mathcal{K}}}) \mid \boldsymbol{v}_{\boldsymbol{\mathcal{K}}} \in \mathcal{K}\}$. We then get the property which we wish to convolve of each neighbour and multiply it by the neighbour's weight, such that $\mathcal{K}_w = \{weight(\boldsymbol{v}) * \mathcal{V}_{property}(\boldsymbol{v}) \mid \boldsymbol{v} \in \mathcal{K}_v \cap \mathcal{V}\}$. The property after convolution is then given by $\mathcal{K}_f(\mathcal{K}_w)$. We denote the convolution of a property of every voxel in $\mathcal{V}$ with $\mathcal{K}$ as $\mathcal{V}_{property,\ \mathcal{K}} = c(\mathcal{V}_{property}, \mathcal{K})$. If the property is left out it is implied to be occupancy.

Voxel hashing

4.1.3 Topological map

A hybrid map representation combining both the topological and metric 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 landmark detection or obstacle avoidance. In the context of this thesis a topological-metric map refers to a 3D representation of an environment containing both a metric occupancy grid map $\mathbb{M}_M$ representing its geometry and a spatial graph $\widetilde{\mathbb{M}_T} = (V, E)$ representing its topology. Each vertex $v \in V$ has an associated variable $m(v)$ representing a subset of the full metric map describing that place, such that $m(v) \subset \mathbb{M}_M$.

4.1.4 Topometric map

Topological maps are a qualitative 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 inspired by the fact that humans are capable of spatial learning despite limited sensory and processing capability and only having partial knowledge of the environment. This is 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 the topology of an environment as a graph G, where vertices V represent n distinctive places v_i and edges E represent the presence of m traversable paths between neighbouring pairs of places $\{v_j, v_k\}$, such that $G = (V, E)$, $V = \{v_i\}_{i=1}^{n}, E = \{\{v_j, v_k\}_i\}_{i=1}^{m}, v_j \in V, v_k \in V$. In the context of indoor mapping the graph is often embedded in 2D or 3D euclidean space as a spatial graph. Given the embedding $f : G \to R^n$, $\widetilde{G} := f(G)$, we denote $\widetilde{G}$ as the spatial graph of G (Kobayashi, 1994). Each vertex in $\widetilde{G}$ has an associated 3D coordinate describing its position in the coordinate frame of the map. If the specific paths between vertices are known they can be associated with the edges in $\widetilde{G}$, otherwise an edge only represents a possible path between two points in space.

4.2 Map Extraction

In the map extraction module we extract topometric maps from each individual partial point cloud map. In this subsection we will discuss the algorithms and data structures used for this purpose.

4.2.1 Overview

The goal of the map extraction step is to transform a partial voxel grid map $\mathcal{V}$ into a topometric map $\mathcal{T}$. It works by first finding a navigation graph $\mathcal{G}_{navigation}$ that connects a subset of the partial map $\mathcal{V}_{navigation} \in \mathcal{V}$, which represents the parts of the map that a hypothetical agent would use to move through the environment, including the floor, stairs and ramps. We then use $\mathcal{G}_{navigation}$ along with $\mathcal{V}$ to segment the map into semantically meaningful rooms based on clusters of visibility, we denote the segmented voxel grid map as $\mathcal{V}_{room}$. After doing so we construct the topological graph of the environment by finding which rooms are adjacent in $\mathcal{G}_{navigation}$. Finally, we fuse the topological map with the segmented room map to construct the topometric map $\mathcal{T}$. Figure 4 shows an overview of our map extraction algorithm, its input, and (intermediate) outputs. In the rest of this subsection we will discuss each of steps in more detail.

4.2.2 Navigable volume

In our first step we extract a navigation graph $\mathcal{G}_{navigation}$. The navigation graph tells us how a theoretical agent in the map's environment would navigate through that environment. In practice, 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.

4.2.3 Convolution

The first step uses voxel convolution with a stick-shaped kernel $\mathcal{K}_{stick}$ (shown in 5) to find all voxels that are unobstructed and are thus candidates for navigation. Each voxel in the kernel has a weight of 1, except the origin voxel which has weight 0. The associated function is summation. Convolving the voxel grid's occupancy property with the stick kernel gives us each voxel's obstruction property, denoted as $\boldsymbol{v}_{obstruction}$ with an obstruction 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, such that $\mathcal{V}_{unobstructed} = \{\boldsymbol{v} \mid \boldsymbol{v} \in convolve(\mathcal{V},\ \mathcal{K}_{stick}),\ \boldsymbol{v}_{obstruction} = 0\}$.

4.2.4 Dilation

The next step of the algorithm is to dilate the unobstructed voxels upwards by a distance d_{dilate}. This connects the unoccupied voxels separated by a small height differences into a connected volume. The value of d_{dilate} depends on the

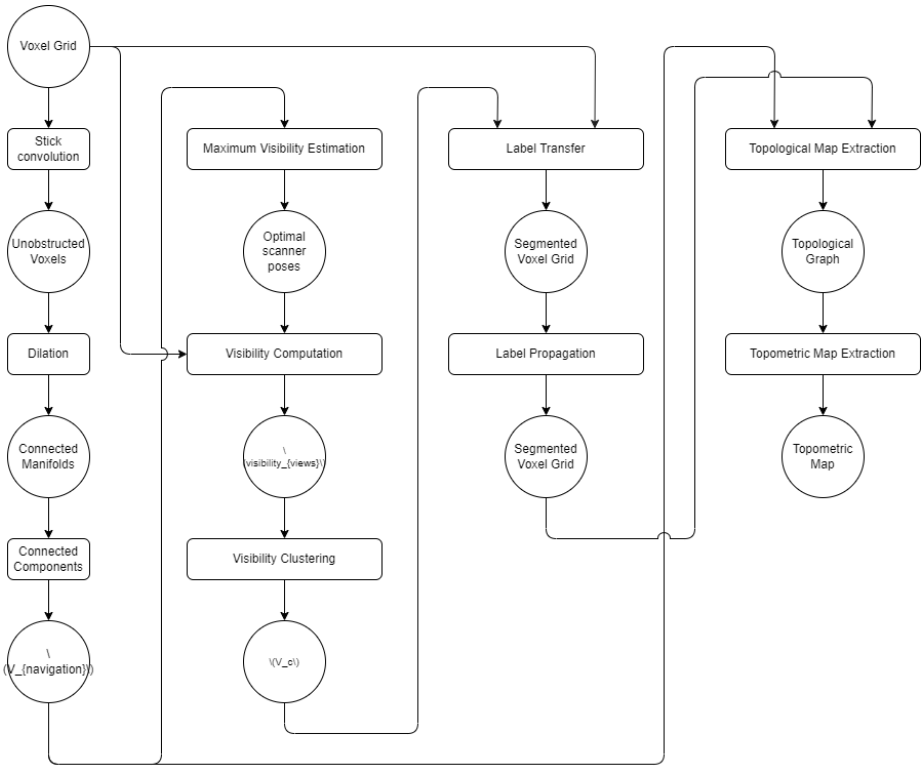

Figure 4: Diagram showing map extraction processes and intermediate data.

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 $\mathcal{V}$.

4.2.5 Connected components

The final step of the algorithm is to split $\mathcal{V}_{dilated}$ into one or more connected components. A connected component $\mathcal{V}_c$ of a voxel grid is a subset of $\mathcal{V}$ where there exists a path between every voxel in $\mathcal{V}_c$. The neighbourhood graph $\mathcal{G}_{\mathcal{V},\ \mathcal{K}} = (V,\ E)$ of $\mathcal{V}$ represents the voxels in $\mathcal{V}$ as nodes. Each node has incident edges towards all other voxels in its neighbourhood, which is defined by a kernel $\mathcal{K}$, such that $V = \mathcal{V},\ E_V = \{(\boldsymbol{v}, \boldsymbol{v_{nb}}) \mid \boldsymbol{v} \in \mathcal{V},\ \boldsymbol{v_{nb}} \in \mathcal{K}_{\boldsymbol{v}}\},\ |E_V| \leq |\mathcal{K}| * |\mathcal{V}|$.

There exists a path between two voxels $\boldsymbol{v_a},\ \boldsymbol{v_b}$ in $\mathcal{V}$ if there exists a path between their corresponding nodes $V_a,\ V_b$ in $\mathcal{G}_{\mathcal{V}}$. We denote the set of all possible paths between two nodes as $paths(V_a,\ V_b)$. A connected component is then defined as $\mathcal{V}_c = \{\boldsymbol{v_a} \mid \boldsymbol{v_a} \in \mathcal{V},\ \boldsymbol{v_b} \in \mathcal{V},\ |paths(V_a, V_b)| \neq 0\}$. We define

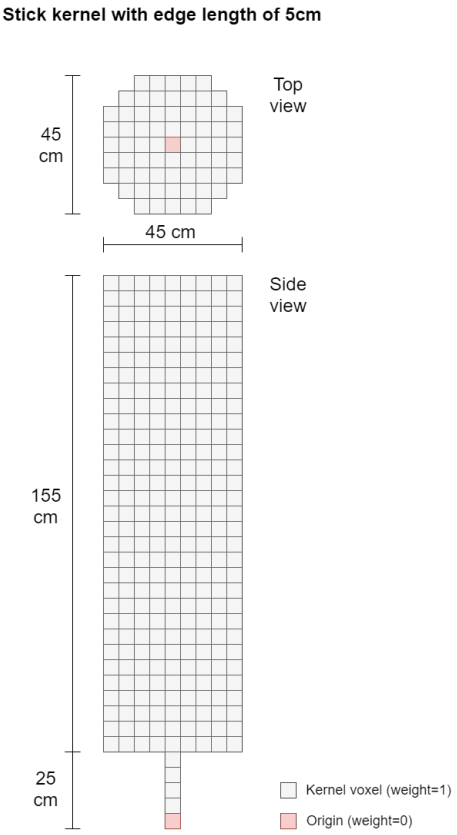

Figure 5: Illustration of stick kernel with top and side views.

the set of all connected components as $\mathcal{V}_{cc} = \{\mathcal{V}_{c,\ i}\}_{i=1}^{n}$. We find the connected components using the following algorithm:

After doing so, we find the connected component in $\mathcal{V}_{cc}$ with the largest amount of voxels $\mathcal{V}_{max}$, which corresponds to the voxels used for navigation. The navigation graph $\mathcal{G}_{navigation}$ is the neighbourhood graph of $\mathcal{V}_{max}$. We denote the intersubsection of the navigation graph's nodes and the whole voxel grid map as $\mathcal{V}_{navigation} = \mathcal{V}_{max} \cap \mathcal{V}$.

4.2.6 Maximum visibility estimation

To compute the isovists necessary for room segmentation it is first necessary to estimate hypothetical scanning positions that maximize the view of the map. We compute these by finding the local maxima of the horizontal distance field of $\mathcal{V}_{navigation}$. The steps to achieve this are as follows.

Algorithm 1 Region growing connected components

Require: Voxel grid $\mathcal{V}$
Require: Connectivity kernel $\mathcal{K}$
Ensure: Connected components $\mathcal{V}_{cc}$
$\mathcal{G}_{\mathcal{V},\,\mathcal{K}} = (V,\ E),\ V \in \mathcal{V}$ ▷ Convert voxel grid to neighbourhood graph using specified kernel
$V_{unvisited} = V$
$\mathcal{V}_{cc} = \{\}$
$i = 0$
while $|V_{visited}| \neq 0$ **do**
  Select random node n from $V_{unvisited}$
  $n_{BFS} = BFS(n)$ ▷ Use breadth-first search to find all nodes connected to n
  Remove n_{BFS} from $V_{unvisited}$
  $\mathcal{G}_i = (n_{BFS},\ E)$
  Add $\mathcal{V}_{c,\,i}$ to $\mathcal{V}_{cc}$
  $i = i + 1$
end while

4.2.7 Horizontal distance field

For each voxel in $\mathcal{V}$ we compute the horizontal Manhattan distance to the nearest boundary voxel. A boundary voxel is a voxel for which 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. Thus, 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 $HDF = \{dist(\boldsymbol{v}) \mid \boldsymbol{v} \in \mathcal{V}\}$. We denote the horizontal distance of a given voxel $\boldsymbol{v}$ as $d_{\boldsymbol{v}}$. We implement this using the following algorithm.

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 $HDF_{max} = \{\boldsymbol{v} \mid dist(\boldsymbol{v}) \geq max\{dist(\boldsymbol{v_r} \mid \boldsymbol{v_r} \in radius(\boldsymbol{v},\ r))\}\}$. 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, to estimate the positions with the optimal view of the map. We denote these positions as $views = \{\boldsymbol{v_c} + (0, h, 0) \mid \boldsymbol{v} \in HDF_{max}\}$. We use the following algorithm to compute $views$.

Algorithm 2 Horizontal distance field

Require: Navigation voxel grid $\mathcal{V}_{navigation}$
Ensure: 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 $convolve(\boldsymbol{v},\ \mathcal{K}_{circle}) = |\mathcal{K}_{circle}|$) **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

Algorithm 3 Horizontal Distance Field Maxima

Require: Navigation voxel grid $\mathcal{V}_{navigation}$
Require: Horizontal distance field HDF
Require: Radius $r \in \mathbb{R}^+$
Require: Scanning height $h \in \mathbb{R}$
Ensure: Estimated optimal views $views \in \mathbb{R}^3$

$views = \{\}$
for each $\boldsymbol{v} \in \mathcal{V}_{navigation}$ **do**
  $neighbourhood = radius(\boldsymbol{v},\ r)$
  if $d_{\boldsymbol{v}} \geq \max\{d_{nb} \mid neighbourhood\}$ **then** ▷ Check if voxel's horizontal distance is equal or greater than the horizontal distance of every voxel in its neighbourhood
    $views = views \cup \{centroid(\boldsymbol{v}) + (0, h, 0)\}$
  end if
end for

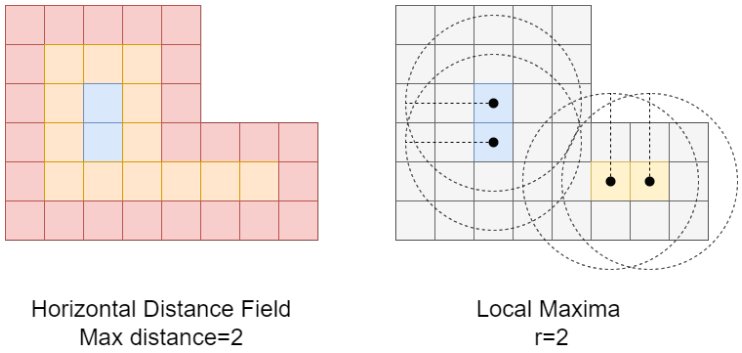

Figure 6: Illustration of horizontal distance field computation and extraction of local maxima.

4.2.8 Visibility

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 $visibility : \mathbb{R},\ \mathbb{Z}^{n\times 3} \mapsto \mathbb{Z}^{m\times 3},\ m \in \mathbb{R},\ n \geq m$. 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. 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. We perform this raycasting operation from every position in *views* towards every voxel within a radius $r_{visibility}$ of that position. Only taking into account voxels 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 $visibility_{views} = \{visibility(\boldsymbol{x}) \mid \boldsymbol{x} \in views\}$. The below pseudocode shows how the visibility computation works.

Algorithm 4 Visibility

Require: Voxel grid $\mathcal{V}$
Require: Origin $o \in \mathbf{R}^3$
Require: Range $r_{visibility} \in \mathbb{R}^+$
Ensure: $visibility \in \mathbb{Z}^3$
$visibility = \{\boldsymbol{v}_c \mid \boldsymbol{v}_c \in radius(o, r_{visibility}) \wedge DDA(\mathcal{V},\ o,\ centroid(\boldsymbol{v}_c)) = \boldsymbol{v}_c\}$

4.2.9 Room segmentation

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

Algorithm 5 DDA (Digital Differential Analyzer)

Require: Voxel grid $\mathcal{V}$
Require: Ray origin $\boldsymbol{o} \in \mathbf{R}^3, \boldsymbol{o} \in [\mathcal{V}_{min}, \mathcal{V}_{max}]$
Require: Ray target $\boldsymbol{t} \in \mathbf{R}^3$
Ensure: $hit \in \mathbb{Z}^3$ ▷ 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

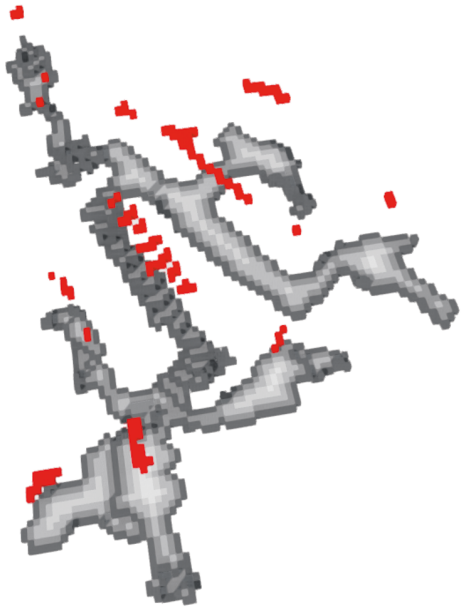

Figure 7: Resulting horizontal distance field of partial map, with resultant optimal view points shown in red.

index, which is given by $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$. Computing the Jaccard index for every combination of visibilities gives us a similarity matrix $S^{n \times n} \in [0,1]$. The similarity matrix is symmetric because $J(A,B) = J(B,A)$. Its diagonals are 1, as $J(A,A) = 1$. An example similarity matrix is showin in figure 8.

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. This means we can treat visibility clustering as a weighted graph clustering problem. To solve this problem we used the Markov Cluster (MCL) algorithm (CITE MCL), 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 $\mathbf{C}_{visibility} = \{c_i\}_{i=1}^{|views|}$, $c_i \in \mathbb{Z}^+$, 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

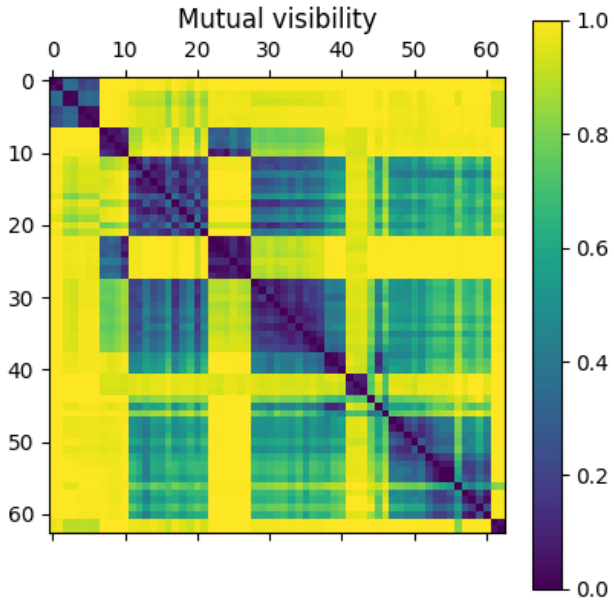

Figure 8: Similarity matrix extracted from set of visibilities. Each value represents the Jaccard index of two visibilities.

$\mathbf{C}_{visibility}$ 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 $\mathcal{V}_c$.

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 room to each voxel in the map. To solve this we assign to each voxel the cluster in which the most visibilities contain that voxel. The result is a mapping from voxels to visibility clusters (which we will from now on refer to as rooms), which we denote as $room : \boldsymbol{v} \mapsto \mathbb{Z}$, such that $room(\boldsymbol{v}) = c,\ c \in \mathbf{C}_{visibility},\ \boldsymbol{v} \in \mathcal{V}$. 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 6 Label propagation

Require: Voxel grid $\mathcal{V}$
Require: Initial labeling $label^{(0)} : \mathbb{Z}^{3+} \mapsto \mathbb{Z}$
Require: Kernel $\mathcal{K}$
Ensure: 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

4.2.10 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 $\mathcal{T} = (, \mathcal{V})$, which consists of a topological graph $= (V, E)$ and a

voxel grid map $\mathcal{V}$. Each node in 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 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.

Algorithm 7 Topology extraction

Require: Voxel grid $\mathcal{V}$
Require: Voxel grid navigation subset $\mathcal{V}_{navigation}$
Require: Room segmentation $room : \boldsymbol{v} \mapsto \mathbb{Z}$
Require: Adjacency kernel $\mathcal{K}_{adjacency}$
Ensure: Topological spatial graph $\widetilde{\mathcal{G}}_{topology} = (V_{topology},\ E_{topology})$
Ensure: Node embedding $f_{node} : V \mapsto \mathbb{R}^3$
  $\widetilde{\mathcal{G}}_{topology} = (\{\},\ \{\})$
  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 $room_{\mathcal{V}} : r \mapsto \mathcal{V}$
  $V_{topology} = \mathbf{V}$
  $f_{node}(v) = centroid(v),\ v \in V_{topology}$
  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

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 $\mathcal{T}_A = (\mathcal{G}_A,\ \mathcal{V}_A)$ and $\mathcal{T}_B = (\mathcal{G}_B,\ \mathcal{V}_B)$. The goal of map matching is to find a mapping $match : v_A \mapsto v_B,\ v_A \in \mathcal{G}_A,\ v_B \in \mathcal{G}_B$ which corresponds to the real world and is robust to differences in coordinate system, resolution and quality between partial maps. To identify matches between nodes we need to be able to compute the similarity between them. To do so, we must first transform each node into a feature vector which represents both the node itself and its relationship to its neighbourhood. The feature vectors of two nodes with similar geometry and a similar neighbourhood should be close to eachother, meaning their Minkowski distance is small. Conversely, the feature vectors of two dissimilar nodes should be far away from eachother. The first step of this process, encoding the node's geometry into a feature vector, is called geometrical feature embedding. The second step, encoding both the geometrical feature embedding of the node itself and of its neighbourhood into a new feature vector is called attributed node embedding. We hypothesize that the attributed node embedding will have better performance for map matching, especially when differences between partial maps are large, because it involves not just the node itself but also its neighbourhood in the similarity measure. This can be compared to human place recognition, where places are identified not just by their appearance but also by their relationship to their context. In this subsection we will discuss multiple algorithms used for geometrical feature embedding and attributed node embedding. We will also discuss how we identify matches between nodes based on their feature vectors.

4.3.1 Geometrical Feature Embedding

Geometrical feature embedding means transforming a geometric object into a feature vector $f \in \mathbb{R}^m$, where m is the dimensionality of the vector. We denote the function that embeds a set of voxels into a feature vector as $embed_{geometry} : \mathbb{Z}^{n\times 3} \mapsto \mathbb{R}^m$. We implement this function using three different approaches, which we discuss below.

4.3.2 Engineered Features

The first approach uses a number of manually engineered features to construct the feature vector from a room's geometry. These features include, for example, the height of the room and its volume. A full list of features and their explanation is given below. More features were tried, but only the ones that were found to contribute to the accuracy of the clustering by trial and error are included here. The features are computed using a point cloud derived from centroids of the occupied voxels of the voxel grid, which we will denote here as $\mathbf{P}$.

Volume The axis aligned bounding box (aabb) of $\mathbf{P}$ is given by the minimum and maximum value along each axis. This results in two 3-dimensional vectors $aabb_{min}$ and $aabb_{max}$. With these vectors we compute the length of each axis of the aabb by computing $\mathbf{l} = aabb_{max} - aabb_{min}$. We then find the volume of the aabb by finding the product of each element of $\mathbf{l}$.

Height Using the same approach as described above we compute the length of each axis of the aabb, $\mathbf{l}$. The height of the point cloud is simply the y-value of $\mathbf{l}$.

Horizontal Area Once again we first compute $\mathbf{l}$. To find the horizontal area of $\mathbf{P}$ we multiply the x- and y-values of $\mathbf{l}$.

Mean distance to centroid To compute the centroid $\mathbf{c}$ of $\mathbf{P}$ we compute the mean value of each axis of all points in $\mathbf{P}$. We then compute the Euclidean distance of each point in $\mathbf{P}$ to $\mathbf{c}$ and compute the mean distance. This metric is closely correlated with an object's volume.

Number of points To compute this value we simply count the number of points in the point cloud. Larger objects will generally contain more points.

Quotient of eigenvalues By using principal component analysis we can determine the 3 eigenvectors and eigenvalues of $\mathbf{P}$, which indicate the directions of maximal variance in the point cloud and the amount of variance along those directions. If all eigenvalues are approximately equal then no direction dominates. We compute if this is the case by finding the quotient of eigenvalues (the first eigenvalue divided by the second and third). If the quotient is close to 1 then no direction dominates.

Ratio of smallest eigenvalue to sum of two largest eigenvalues We take the two largest eigenvalues and find their sum, then we divide the smallest eigenvalue by it. Objects for which the largest two eigenvalues are much larger than the smallest eigenvector have a single direction that is non-dominant.

Verticality of largest eigenvector We take the largest eigenvector, normalize it, and then find the dot product of the eigenvector and the unit vector in the z-direction. This gives us the degree to which the point cloud is vertically aligned. If the largest eigenvalue is non-vertical then the object is mostly horizontal, which is the case for fences, buildings and cars. If the largest eigenvalue is vertical then the object is vertical, which is the case for poles and trees.

Roughness For each point we find their n nearest neighbours. We then fit a plane through the point's neighbourhood, we do this by finding the eigenvectors of the neighbourhood. The smallest eigenvector gives us the normal vector of

the neighbourhood, which along with the neighbourhood's centroid gives us the best fit plane. We then determine the sum distance of each point in the neighbourhood in the plane. The roughness of the point cloud is then given by the mean of the sum distance of each point's neighbourhood to its best fit plane.

4.3.3 ShapeDNA

4.3.4 Deep Learning

Another approach to geometrical feature embedding uses deep learning. This works by using a pretrained model, used for segmentation of objects in indoor environments for example, and using the output of the last hidden layer as the feature vector. We use two different network architectures and models to achieve this, PointNet and DGCNN, which we will describe below.

PointNet

DGCNN

4.3.5 Attributed Node Embedding

Attributed node embedding aims to find a feature embedding for each node in a graph that uses both an attribute of the node, in our case a geometrical feature embedding, and the node's relationship to the rest of the graph. We denote the function that embeds a node's attribute f_{attr} and its graph into a feature vector as $embed_{node} : \mathbb{R}^m,\ \mathcal{G} \mapsto \mathbb{R}^m$, such that $embed_{node}(f_{attr},\ \mathcal{G}_{attr}) = f_{node}$. Finding the attributed node embedding of a node n in the topometric map $\mathcal{T}$ with topological graph $\mathcal{G}_T$ is then equal to computing $f_{node} = embed_{node}(embed_{geometry}(n),\ \mathcal{G}_T)$.

We use a number of different algorithms to solve this problem in our research, which we will describe below.

SINE

MUSAE

FEATHER-N

4.3.6 Map Matching

The above steps are applied to both partial maps. This gives us two sets of feature vectors $\mathcal{E}_A,\ \mathcal{E}_B$ that represent the partial maps' attributed node embedding. Our goal is to find a injection between the elements of both sets. To do so, we first find the Cartesian product $\mathcal{E}_{AB} = \mathcal{E}_A \times \mathcal{E}_B = \{(a, b) \mid a \in \mathcal{E}_A,\ b \in \mathcal{E}_B\}$. We then compute the Euclidean distance between every element in $\mathcal{E}_{AB}$, such that $\mathcal{D}_{AB} = \{\sqrt{(a-b)^2} \mid (a, b) \in \mathcal{E}_{AB}\}$. $\mathcal{D}_{AB}$ describes the pairwise distance

between each combination of nodes in the partial maps. To extract an injection between the two sets of nodes from this we use the following algorithm:

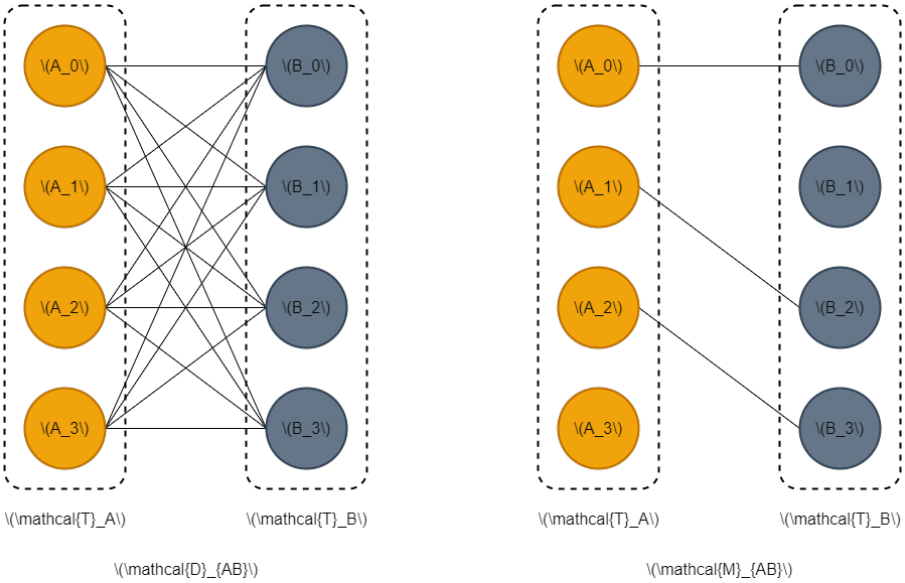

Figure 9: Conversion of similarity between all pairs of rooms to injective non-surjective mapping between rooms.

4.4 Map Fusion

4.4.1 Registration

4.4.2 Transform Hypothesis Clustering

4.4.3 Topological Fusion

5 Implementation Details

5.1 External Libraries

5.2 GPU Acceleration

6 Results

6.1 Map Extraction

6.2 Map Matching

6.3 Map Merging

7 Discussion

7.1 Performance

7.2 Future Works

References

Andersone. (2019, August). Heterogeneous Map Merging: State of the Art. *Robotics*, *8*(3), 74. Retrieved 2021-07-16, from https://www.mdpi.com/2218-6581/8/3/74 doi: 10.3390/robotics8030074

Bonanni, T. M., Della Corte, B., & Grisetti, G. (2017, April). 3-D Map Merging on Pose Graphs. *IEEE Robotics and Automation Letters*, *2*(2), 1031–1038. Retrieved 2021-11-09, from https://ieeexplore.ieee.org/document/7822998/ doi: 10.1109/LRA.2017.2655139

Dudek, G., Jenkin, M., Milios, E., & Wilkes, D. (1998). Topological Exploration With Multiple Robots. , 6.

Gholami Shahbandi, S., & Magnusson, M. (2019, June). 2D map alignment with region decomposition. *Autonomous Robots*, *43*(5), 1117–1136. Retrieved 2021-11-09, from http://link.springer.com/10.1007/s10514-018-9785-7 doi: 10.1007/s10514-018-9785-7

Gorte, B., Zlatanova, S., & Fadli, F. (2019, May). NAVIGATION IN INDOOR VOXEL MODELS. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *IV-2/W5*, 279–283. Retrieved 2021-12-06, from https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-2-W5/279/2019/ doi: 10.5194/isprs-annals-IV-2-W5-279-2019

He, Z., Sun, H., Hou, J., Ha, Y., & Schwertfeger, S. (2021, June). Hierarchical topometric representation of 3D robotic maps. *Autonomous Robots*, *45*(5), 755–771. Retrieved 2021-11-09, from https://link.springer.com/10.1007/s10514-021-09991-8 doi: 10.1007/s10514-021-09991-8

Hermann, M., Pentek, T., & Otto, B. (2016, January). Design Principles for Industrie 4.0 Scenarios. In *2016 49th Hawaii International Conference on System Sciences (HICSS)* (pp. 3928–3937). (ISSN: 1530-1605) doi: 10.1109/HICSS.2016.488

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

Kobayashi, K. (1994, January). On the spatial graph. *Kodai Mathematical Journal*, *17*(3). Retrieved 2022-01-21, from https://projecteuclid.org/journals/kodai-mathematical-journal/volume-17/issue-3/On-the-spa
doi: 10.2996/kmj/1138040046

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.

Kuipers, B., & Byun, Y.-T. (1991, November). A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Robotics and Autonomous Systems*, *8*(1), 47–63. Retrieved 2022-01-10, from https://www.sciencedirect.com/science/article/pii/092188909190014C doi: 10.1016/0921-8890(91)90014-C

Li, Y., & Olson, E. B. (2010, November). A General Purpose Feature Extractor for Light Detection and Ranging Data. *Sensors*, *10*(11), 10356–10375. Retrieved 2022-01-24, from https://www.mdpi.com/1424-8220/10/11/10356 (Number: 11 Publisher: Molecular Diversity Preservation International) doi: 10.3390/s101110356

Ochmann, S., Vock, R., Wessel, R., & Klein, R. (2014). Towards the Extraction of Hierarchical Building Descriptions from 3D Indoor Scans. , 8.

Rusinkiewicz, S., & Levoy, M. (2001, May). Efficient variants of the ICP algorithm. In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling* (pp. 145–152). doi: 10.1109/IM.2001.924423

Serafin, J., & Grisetti, G. (2015, September). NICP: Dense normal based point cloud registration. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 742–749). doi: 10.1109/IROS.2015.7353455

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

Tomatis, N., Nourbakhsh, I., & Siegwart, R. (2003, July). Hybrid simultaneous localization and map building: a natural integration of topological and metric. *Robotics and Autonomous Systems*, *44*(1), 3–14. Retrieved 2021-12-20, from https://linkinghub.elsevier.com/retrieve/pii/S092188900300006X doi: 10.1016/S0921-8890(03)00006-X

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

Yang, J., Cao, Z., & Zhang, Q. (2016, June). A fast and robust local descriptor for 3D point cloud registration. *Information Sciences*, *346-347*, 163–179. Retrieved 2022-01-18, from

https://www.sciencedirect.com/science/article/pii/S0020025516300378 doi: 10.1016/j.ins.2016.01.095

Yu, S., Fu, C., Gostar, A. K., & Hu, M. (2020, December). A Review on Map-Merging Methods for Typical Map Types in Multiple-Ground-Robot SLAM Solutions. *Sensors*, *20*(23), 6988. Retrieved 2021-07-12, from https://www.mdpi.com/1424-8220/20/23/6988 doi: 10.3390/s20236988