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

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. Collaborative mapping of indoor environments is especially challenging, because 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). Partial maps might represent 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 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. 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

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

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

In this section we will describe our methodology for answering our research questions. We can divide our methodology into three major components: map extraction, map matching and map fusion, which correspond with the three sub-research questions. This section follows this division, with an added subsection describing the different map representations that are used in-depth. In an overview, our methodology can be summarized as follows.

Map extraction For each input partial map, in the form of a point cloud, we create a voxel grid with equal cell size. Within these voxel grids, we detect which voxels could feasible be used to navigate through the environment. Using this information we segment the voxel grids into submaps, which we refer to as rooms, by finding areas with many common viewpoints. By combining the room submaps with the navigable voxels we extract a topometric representation from each partial map containing both the room submaps and their topological relationships as a graph.

Map matching We then identify matches between rooms to detect overlapping areas between the partial maps. We do this based on the rooms' geometric features and those of their context, the rooms that are n steps away in the topological graph. We further improve our matching by generating multiple matching hypotheses along their topological graphs and selecting the one with the best quality.

Map fusion Afterwards, we use the matches to find a rigid transform between the partial maps that brings the overlapping areas into alignment. Finally, we fuse the topological graphs and the room geometry into a single global map.

Figure 3 shows the steps described above as a flowchart. Refer to the specific subsections for each step for a further description of the algorithms and notation.

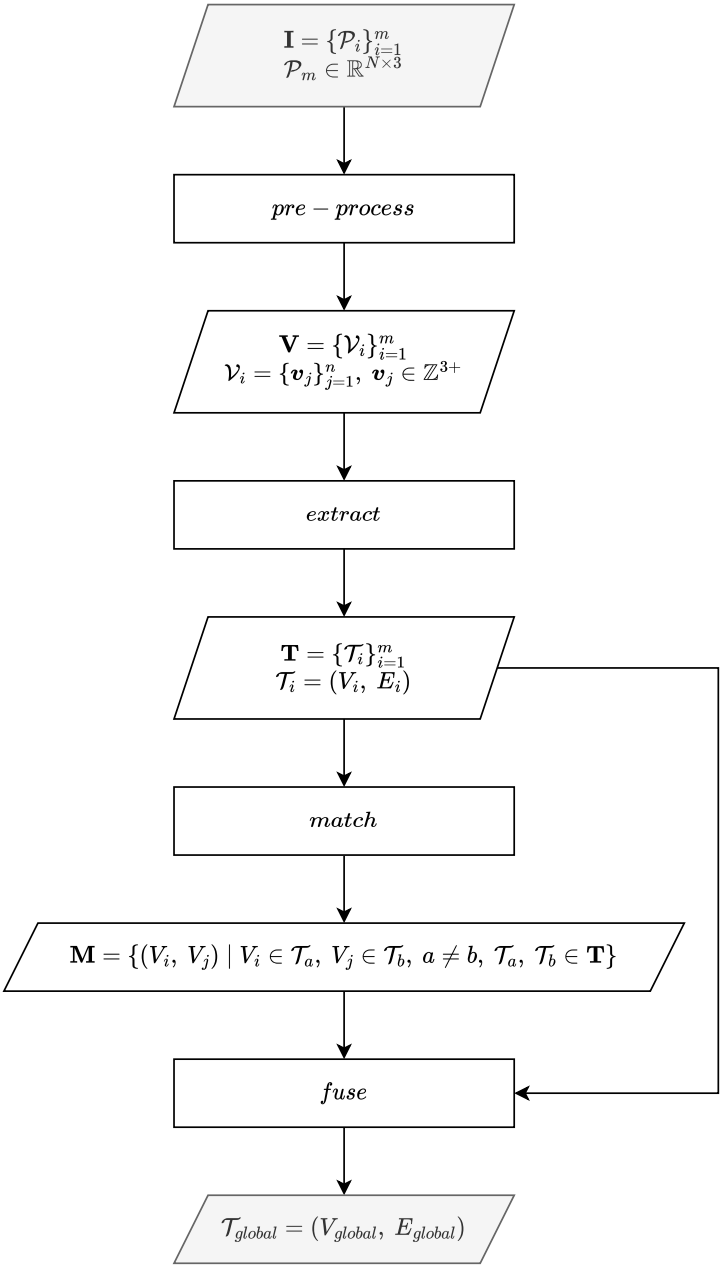

Figure 3: Diagram showing overview of methodology.

4.1 Map Representations

In this section we will give a description of the different kinds of 3D 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.

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. 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 $\boldsymbol{v_{V_{min}}} = (0, 0, 0)$ 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, where // denotes integer division.

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

$$\boldsymbol{v}_{min} = V_{min} + \boldsymbol{v} * e_l \tag{3}$$

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

The minimum and maximum bounds of this subvolume are given by equation 3 and 4. The voxel's centroid is given by 5. Given a point $\boldsymbol{p}$ within the bounds of V, the corresponding voxel is given by equation 6 .

$$\boldsymbol{v_c} = (\boldsymbol{v_{min}} + \boldsymbol{v_{max}}) * 0.5 \tag{5}$$

$$\boldsymbol{v_p} = (\boldsymbol{p} - V_{min}) \text{ // } e_l \tag{6}$$

A property of a voxel is given by equation 7. We define a property as an m by n real matrix for the purposes of this research. However, in reality a property

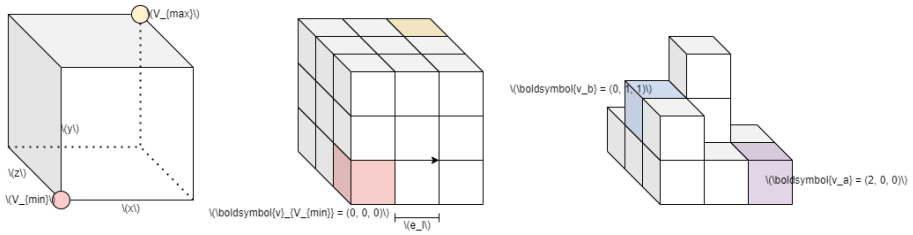

Figure 4: A voxel grid and its components.

could be any value. For example, we denote the property *occupied*, which can be either 1 or 0, of a voxel $\boldsymbol{v}$ as $V_{occupied}(\boldsymbol{v}) = \boldsymbol{v}_{occupied} = (1) \mid (0)$

$$\mathcal{V}_{property} : \mathbb{Z}^{3+} \mapsto \mathbb{R}^{m*n},\ \mathcal{V}_{property}(\boldsymbol{v}) = \boldsymbol{v}_{property} \tag{7}$$

We define a voxel grid as a set of occupied voxels with an associated bounded volume $V_{\mathcal{V}}$ and edge length e_l, as shown in equation 8. Figure 4 shows an example voxel grid and its components.

$$\mathcal{V} = \{\boldsymbol{v}_i\}_{i=1}^{n},\ n \in [1, \prod \boldsymbol{v}_{\boldsymbol{V_{max}}}] \tag{8}$$

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 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 representation of the voxel's coordinates into a single binary string which is interpreted as a positive integer. This is a Morton code. The ordered vector of Morton codes gives us the Morton order. We denote the Morton order of a voxel grid according to equation 9

$$M_{\mathcal{V}} = \{m_i\}_{i=1}^{|\mathcal{V}|},\ m_i < m_{i+1},\ m_i \in \mathbb{Z}^{+} \tag{9}$$

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, 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 follows.

$$radius : \mathbb{Z}^{3+},\ \mathbb{R} \mapsto \mathbb{Z}^{n\times 3} \tag{10}$$

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 over its neighbours.

$$\mathcal{K}_f : \mathcal{K}_w \mapsto \mathbb{R}^{m*n} \tag{11}$$

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 and an origin voxel.

$$weight : \mathbb{Z}^{3+} \mapsto \mathbb{R},\ v \in \mathcal{K} \tag{12}$$

$$\boldsymbol{o}_{\mathcal{K}} \in \mathbb{Z}^3 \tag{13}$$

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

$$\mathcal{K}_{\boldsymbol{v}} = \{\boldsymbol{v}_{\boldsymbol{\mathcal{K}}} + (\boldsymbol{v} - \boldsymbol{o}_{\boldsymbol{\mathcal{K}}}) \mid \boldsymbol{v}_{\boldsymbol{\mathcal{K}}} \in \mathcal{K}\} \tag{14}$$

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

$$\mathcal{K}_w = \{weight(\boldsymbol{v})\mathcal{V}_{property}(\boldsymbol{v}) \mid \boldsymbol{v} \in \mathcal{K}_v \cap \mathcal{V}\} \tag{15}$$

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 follows.

$$\mathcal{V}_{property,\ \mathcal{K}} = c(\mathcal{V}_{property}, \mathcal{K}) \tag{16}$$

If the property is left out it is implied to be occupancy.

4.1.3 Topological 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 nodes N represent distinctive places n and edges E represent the presence of a

path between neighbouring pairs of places (v_j, v_k) that does not pass through any other places.

$$G = (N, E) \tag{17}$$

$$N = \{n_i\}_{i=1}^{k} \tag{18}$$

$$E = \{(n_j, n_k)_i\}_{i=1}^{m}, n_j \in N,\ n_k \in N \tag{19}$$

In the context of indoor mapping the graph is often embedded in 2D or 3D euclidean space as a spatial graph. We denote he spatial graph of G as $\widetilde{G}$. Each vertex in $\widetilde{G}$ has an associated 3D coordinate describing its position in the coordinate frame of the map.

4.1.4 Topometric 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 topometric map refers to a 3D representation of an environment containing both a metric voxel grid map representing its geometry and a topological graph representing the adjacency relationship between distinctive places.

$$\mathcal{T} = (\mathcal{V},\ G),\ \mathcal{V} = \{\boldsymbol{v}_i\}_{i=1}^{n} \tag{20}$$

$$G = (N,\ E),\ N = \{n_i\}_{i=1}^{k},\ n \subset \mathcal{V} \tag{21}$$

Each node $n \in N$ has an associated variable $m(v)$ representing a subset of the full metric map describing that place.

4.2 Map Extraction

The goal of the topometric map extraction step is to transform a partial voxel grid map of an indoor environment, denoted by $\mathcal{V}$, into a topometric map, denoted by $\mathcal{T}$. The algorithm should be robust the missing data and work across a wide range of data resolution. In this section we propose an algorithm to achieve this goal. In an overview, it works as follows.

We first extract a navigation graph $\mathcal{G}_{navigation}$, a graph connecting all of the voxels which a hypothetical agent could use to move through the environment, from the partial voxel grid map $\mathcal{V}$. 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 divide indoor environments into rooms. As such, we refer to the submaps of $\mathcal{V}$ when segmented using visibility clustering as 'rooms'. Afterwards, we construct the topological graph of the environment, denoted by , with edges representing whether it is possible to navigate between two rooms without passing through another. Finally, we fuse the topological map with the segmented map to construct the topometric map $\mathcal{T}$.

Figure 5 shows an overview of our map extraction algorithm, its input, and intermediate outputs. In the rest of this subsection we will the algorithm in more detail.

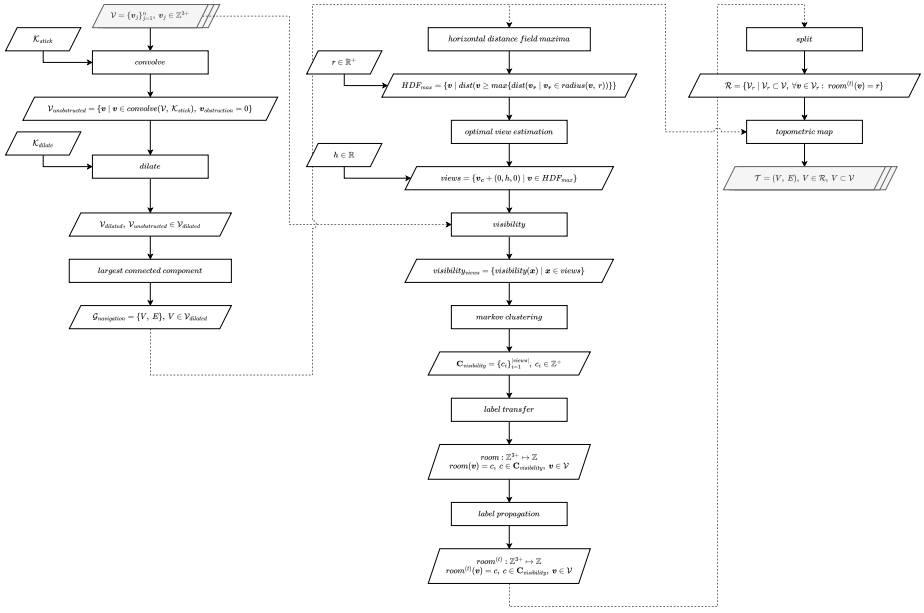

Figure 5: Diagram showing map extraction processes and intermediate data.

4.2.1 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, 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.

4.2.2 Convolution

The first step uses voxel convolution with a stick-shaped kernel $\mathcal{K}_{stick}$ (shown in 6) 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 equation 22 follows.

$$\mathcal{V}_{unobstructed} = \{\boldsymbol{v} \mid \boldsymbol{v} \in convolve(\mathcal{V},\ \mathcal{K}_{stick}),\ \boldsymbol{v}_{obstruction} = 0\} \tag{22}$$

4.2.3 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 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 $\mathcal{V}$.

4.2.4 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 $\mathcal{G}_{\mathcal{V},\ \mathcal{K}} = (N,\ E)$ 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

$$N = \mathcal{V} \tag{23}$$

$$E_V = \{(\boldsymbol{v}, \boldsymbol{v_{nb}}) \mid \boldsymbol{v} \in \mathcal{V},\ \boldsymbol{v_{nb}} \in \mathcal{K}_{\boldsymbol{v}}\},\ |E_V| \leq |\mathcal{K}| * |\mathcal{V}| \tag{24}$$

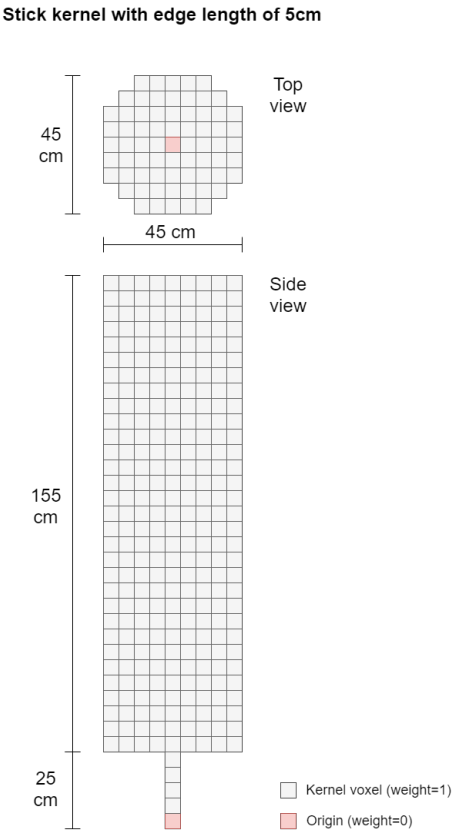

Figure 6: Illustration of stick kernel with top and side views.

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 $N_a,\ N_b$ in $\mathcal{G}_{\mathcal{V}}$. We denote the set of all possible paths between two nodes as $paths(N_a,\ N_b)$. We denote a connected component containing voxel $\boldsymbol{v_a}$ and all voxels conntected to it as:

$$\mathcal{V}_a = \{\boldsymbol{v_b} \mid \boldsymbol{v_a} \in \mathcal{V},\ \boldsymbol{v_b} \in \mathcal{V},\ |paths(N_a, N_b)| \neq 0\} \tag{25}$$

We define 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 $\mathcal{V}_{navigation} = \mathcal{V}_{max} \cap \mathcal{V}$.

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

4.2.5 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.

4.2.6 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

$$HDF = \{dist(\boldsymbol{v}) \mid \boldsymbol{v} \in \mathcal{V} \tag{26}$$

$$HDF_{max} = \{\boldsymbol{v} \mid dist(\boldsymbol{v}) \geq max\{dist(\boldsymbol{v_r} \mid \boldsymbol{v_r} \in radius(\boldsymbol{v},\ r))\}\} \tag{27}$$

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

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

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 27 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, to estimate the positions with the optimal view of the map. We denote these positions as

$$views = \{\boldsymbol{v_c} + (0, h, 0) \mid \boldsymbol{v} \in HDF_{max}\} \tag{28}$$

We use the following algorithm to compute $views$.

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

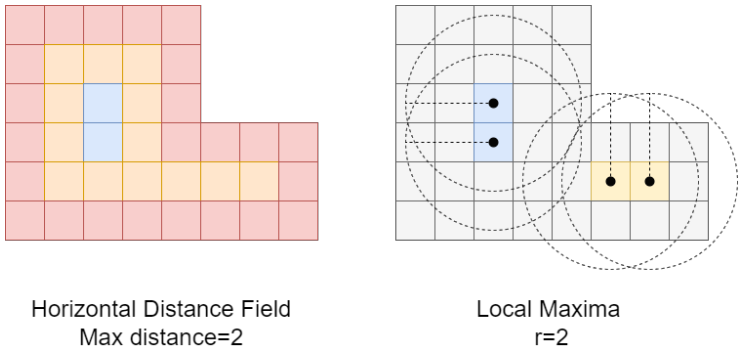

Figure 7: Illustration of horizontal distance field computation and extraction of local maxima.

4.2.7 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

$$visibility : \mathbb{R},\ \mathbb{Z}^{n\times 3} \mapsto \mathbb{Z}^{m\times 3},\ m \in \mathbb{R},\ n \geq m \tag{29}$$

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. Figure 9 shows a 2D representation of voxel raycasting.

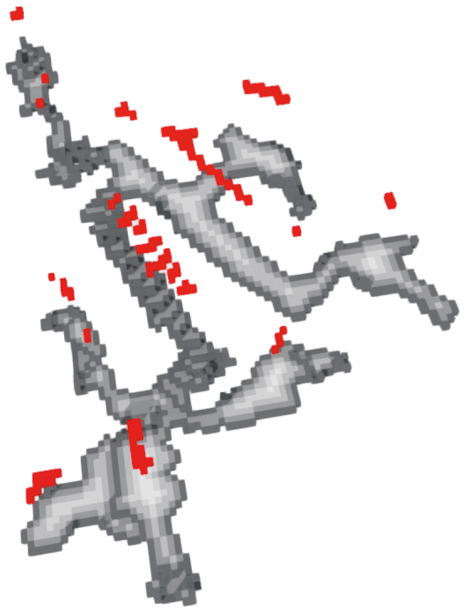

Figure 8: Resulting horizontal distance field of partial map, with resultant optimal view points shown in red.

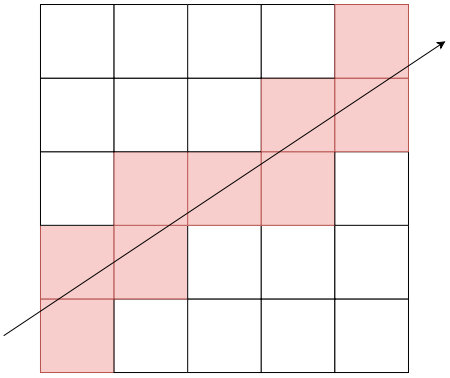

Figure 9: 2D representation of voxel raycasting.

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

$$visibility_{views} = \{visibility(\boldsymbol{x}) \mid \boldsymbol{x} \in views\} \tag{30}$$

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: $visibility \in \mathbb{Z}^3$
$visibility = \{\boldsymbol{v}_c \mid \boldsymbol{v}_c \in radius(o, r_{visibility}) \wedge DDA(\mathcal{V},\ o,\ centroid(\boldsymbol{v}_c)) = \boldsymbol{v}_c\}$

4.2.8 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 index, which is given by equation 31.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{31}$$

$$J(A, B) = J(B, A) \tag{32}$$

$$J(A, A) = 1 \tag{33}$$

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 of equation 32. Its diagonals are 1, as follows from equation 33. An example similarity matrix and its corresponding graph are shown in figure 10 and 11.

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

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

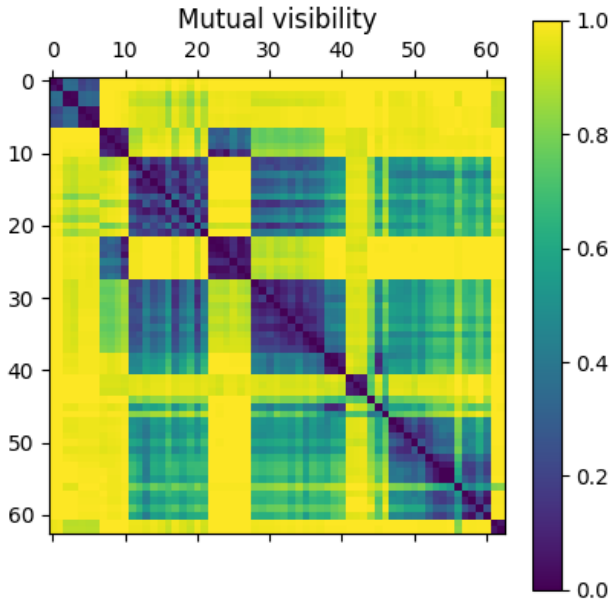

Figure 10: Similarity matrix extracted from set of visibilities. Each value represents the Jaccard index of two visibilities.

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

$$\mathbf{C}_{visibility} = \{c_i\}_{i=1}^{|views|},\ c_i \in \mathbb{Z}^+ \tag{34}$$

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 $\mathcal{V}_c$.

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

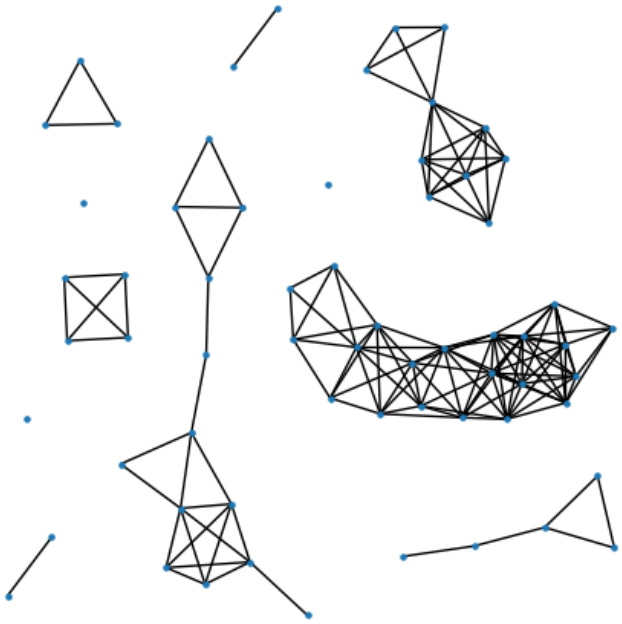

Figure 11: Graph representation of the similarity matrix, edges under a threshold similarity are removed. Nodes represent visibilities.

$$room : \boldsymbol{v} \mapsto \mathbb{Z} \tag{35}$$

$$room(\boldsymbol{v}) = c,\ c \in \mathbf{C}_{visibility},\ \boldsymbol{v} \in \mathcal{V} \tag{36}$$

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.

4.2.9 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

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) ≠ label^(t+1))   ▷ Keep iterating until labels stop changing
    for each ( do v ∈ V)
        L = {label^(t)(v_nb) | v_nb ∈ neighbours(v, K)}4
        l_max = argmax_l |{l | l ∈ L}|   ▷ Most common label in neighbourhood
        l_current = label^(t)(v)                ▷ Label of current voxel
        if |{l | l ∈ L ∧ l = l_max}| > |{l | l ∈ L ∧ l = l_current| then
            label^(t+1)(v) = l_max
        else
            label^(t+1)(v) = label^(t)(v)
        end if
    end for
    t = t + 1          ▷ Use propagated labeling as input for next iteration
end while
```

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.

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

$$\mathcal{T}_a = (\mathcal{G}_a,\ \mathcal{V}_a),\ \mathcal{G}_a = (N_a, E_a) \tag{37}$$

$$\mathcal{T}_b = (\mathcal{G}_b,\ \mathcal{V}_b),\ \mathcal{G}_b = (N_b, E_b) \tag{38}$$

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

$$match : n_a \mapsto n_b,\ n_a \in N_a,\ n_b \in N_b \mid n_b = null \tag{39}$$

$$match(n_a) = \begin{cases} n_b \\ null \end{cases} \tag{40}$$

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 distance in feature space 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 geometric feature embedding. The second step, encoding both the geometric feature embedding of the node itself and of its neighbourhood into a new feature vector is called attributed node embedding. Figure 12 shows an overview of the steps described above.

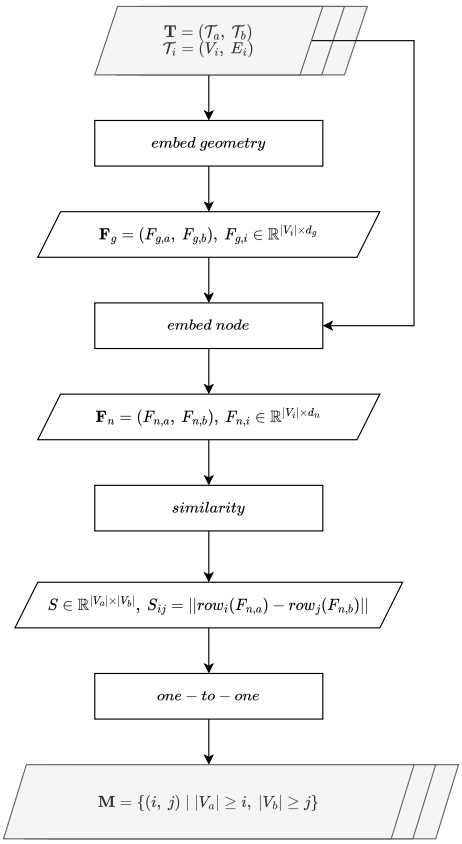

Figure 12: Diagram showing map matching methodology.

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

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

$$f_{geometry} \in \mathbb{R}^m \tag{41}$$

$$embed_{geometry} : \mathbb{Z}^{n\times 3} \mapsto \mathbb{R}^m \tag{42}$$

$$embed_{geometry}(n) = {}^{n}f_{geometry} \tag{43}$$

We denote the function that embeds a set of voxels into a feature vector as in equation 42. 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 Spectral Features

Another approach to feature embedding uses the first n sorted nonzero eigenvalues of the adjacency or Laplacian matrix of a graph. In our case, the graph refers to the neighbourhood graph of the voxel grid associated with each node in the topometric map. By adjusting the size of the kernel used to construct the neighbourhood graph we can adjust the number of edges, which influences the resulting embedding. We use singular value decomposition to find the eigenvalues.

4.3.4 Deep Learning

Our final approach to feature embedding is deep learning. In this approach, the geometry of the nodes is fed into a neural network that is trained on a specific task, such as semantic segmentation or classification, and the intermediate output of a hidden layer is used as a feature vector. We use an architecture called Point Completion Network (PCN), which is an autoencoder used for the specific task of completing incomplete point clouds. Autoencoders consist of two components: an encoder and a decoder. The encoder component is responsible for reducing the dimensionality of the input data into a single n-dimensional feature vector which captures the input's defining characteristics. The decoder is responsible for recovering the input data from the encoder's output as accurately as possible. For our purposes, only the output of the encoder is required. The advantage of using an autoencoder network is that they are trained in an unsupervised manner, so no manually labelled data is required. We train our

model on the Stanford 3D Indoor Scene dataset (see results section). To generate incomplete views we compute visibilities from one or multiple random poses in each room.

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 l-dimensional feature vector.

$$embed_{node} : \mathbb{R}^m,\ \mathcal{G} \mapsto \mathbb{R}^l \tag{44}$$

$$embed_{node}({}^n f_{attr},\ \mathcal{G}) = {}^n_{node} f_{attr},\ n \in N \tag{45}$$

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

$${}^n_{node} f_{geometry} = embed_{node}({}^n f_{geometry},\ \mathcal{G}_T) \tag{46}$$

We compute the attributed node embedding using graph convolution. This means that for every node, we find its adjacent nodes and add their feature vectors weighted by a factor w to the origin node's feature vector. If we denote the adjacent nodes of v in graph G as

$$N_G(v) = \{v_i\}_{i=1}^k \tag{47}$$

Then each node's feature vector after graph convolution becomes

$$v^{t+1} = v^t + \sum_{i=1}^{k} w N_G(v^t)_i \tag{48}$$

By repeating this step the embedding of a node's neighbourhood is integrated in each node's embedding, making nodes that have a similar neighbourhood more similar and vice versa. Increasing the number of iterations also increases the distance at which a neighbour influences a node's embedding. Changing the value of w increases the influence of neighbours.

4.3.6 Map Matching

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

$$\mathcal{E}_{AB} = \mathcal{E}_A \times \mathcal{E}_B = \{(a, b) \mid a \in \mathcal{E}_A,\ b \in \mathcal{E}_B\} \tag{49}$$

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

$$\mathbf{C} \in \mathbb{R}^{|V_a| \times |V_b|} \tag{50}$$

$$\mathbf{C}_{ij} = ||a - b||,\ (a,\ b) \in \mathcal{E}_{AB},\ a = \mathcal{E}_{A,\ i},\ b = \mathcal{E}_{B,\ j},\ \mathbf{C}_{ij} \in \mathbb{R}^{+} \tag{51}$$

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

$$\mathbf{M} = \{(i,\ j,\ d) \mid |V_a| \geq i,\ |V_b| \geq j,\ d = \mathbf{C}_{ij}\} \tag{52}$$

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 match

$$(v_i,\ v_j) = (V_{A,\ i},\ V_{B,\ j}),\ (i,\ j) \in \mathbf{M} \tag{53}$$

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 coarse registration between the centroids of the geometry of the identified matches at every step of the hypothesis growing using least squares adjustment. 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.

The hypothesis growing step produces multiple hypotheses, one for every initial matching. As described in Huang (Huang & Beevers, 2005) we then cluster the hypotheses by similarity and compatibility. Two hypotheses are

compatible if there are no contradicting matches between them. They are similar if the distance between their coarse registration computed during the hypothesis growing process is small. If two hypotheses are incompatible their distance is set to infinite. We use the OPTICS algorithm to cluster hypotheses. After clustering the hypotheses multiple hypotheses might still remain. We select the hypothesis with the largest number of matches as the most likely hypothesis. If there are multiple hypotheses with an equal largest number of matches then the hypothesis with the lowest mean distance in feature space between matches is selected.

4.4 Map Fusion

The final step of the map merging process is map fusion. In the case of topometric maps this means fusion at the geometric and topological level. We arbitrarily designate one partial map as the source and the other as the target. The goal of geometric fusion is to find a rigid transformation that aligns the source map with the target.

At the topological level, we first merge the nodes of the source map with their corresponding matches in the target map. If two adjacent nodes in one map match with two other adjacent nodes in the other partial map then their edges are also merged. Afterwards, a transformation between the geometry (a point cloud derived from the voxel grid's centroids) of each matching node is computed using a fine registration approach. We use two different algorithms two do this: iterative closest point (ICP) and deep closest point (DCP). Figure 13 shows an overview of our map fusion methodology.

4.4.1 Geometric fusion

Iterative closest point The iterative closest point (ICP) algorithm finds a transformation between two point clouds by iteratively aligning n random points from the source point cloud to the n points that are closest to them in the target point cloud using least squares adjustment. This approach is sensitive to local minima but can often lead to good results if enough data is available. Its simple nature also makes it easy to modify. For example, constraining the registration to only allow rotation around a single axis is a matter of changing the equations used in the least square adjustment step. This

Deep closest point Deep closest point is a variation on the ICP algorithm which uses deep learning to identify matching points instead of using the closest points. This approach is less sensitive to local minima and data quality. However, constraining the registration is difficult as it requires retraining the network.

After finding a transformation between every match outliers that represent an incorrect registration are removed. We then find the mean of the remaining transformations and apply this to the source map. The result is a global topometric map $\mathcal{T}_{global}$.

4.4.2 Topological fusion

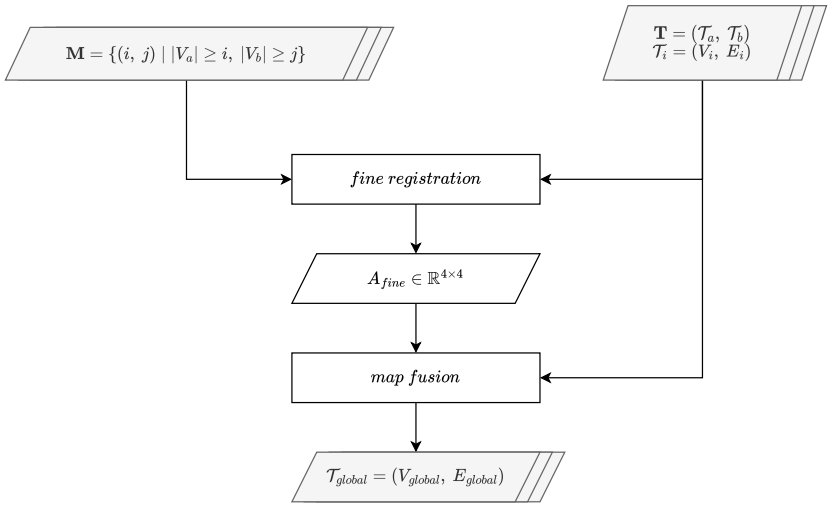

Figure 13: Diagram showing overview of map fusion methodology.

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 14 shows an example global map with simulated viewpoints. Figure 15 and 16 show two simulated partial maps created from a single map.

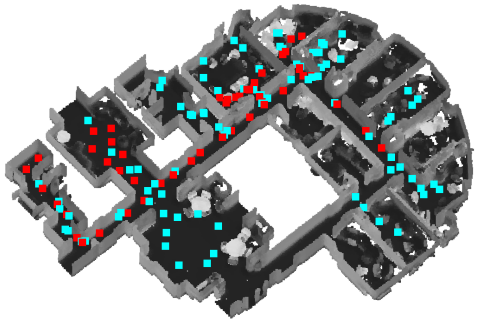

Figure 14: Global map with simulated viewpoints. Each coloured dot represents a viewpoint, the union of all views with the same color forms a partial map.

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.

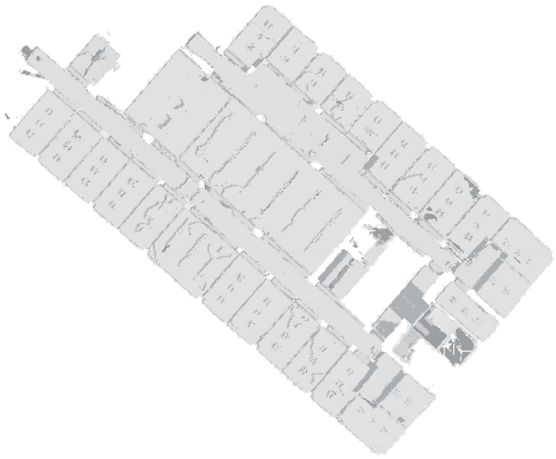

Figure 15: First simulated partial map extracted from S3DIS area 1 dataset.

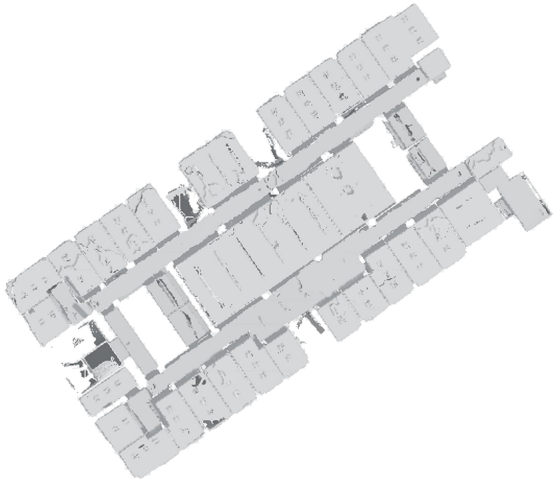

Figure 16: Second simulated partial map extracted from S3DIS area 1 dataset.

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).

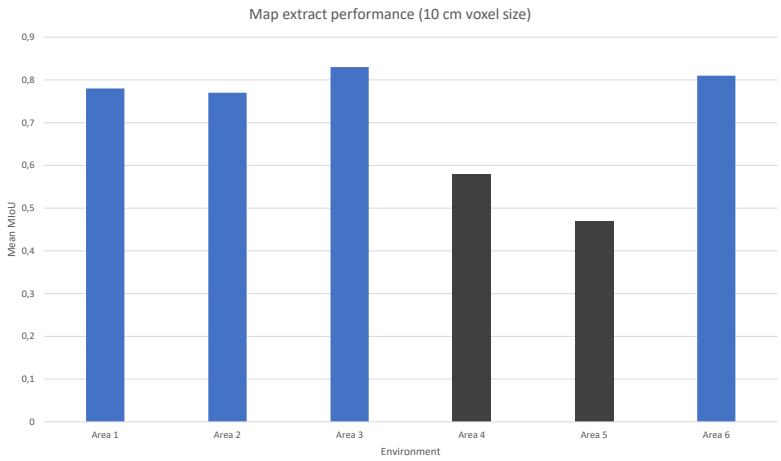

Figure 17: First topometric map extracted from S3DIS area 1 dataset.

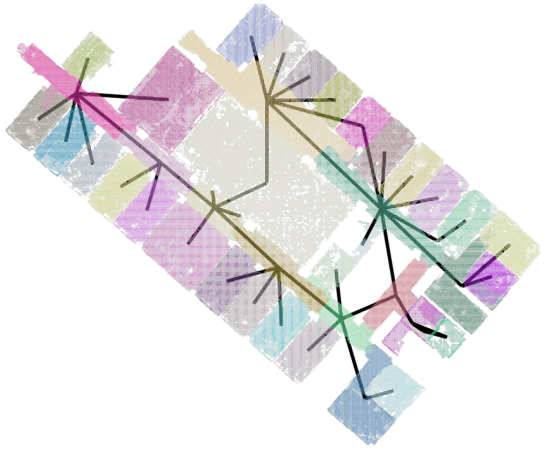

Figure 18: First topometric map extracted from S3DIS area 1 dataset.

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.

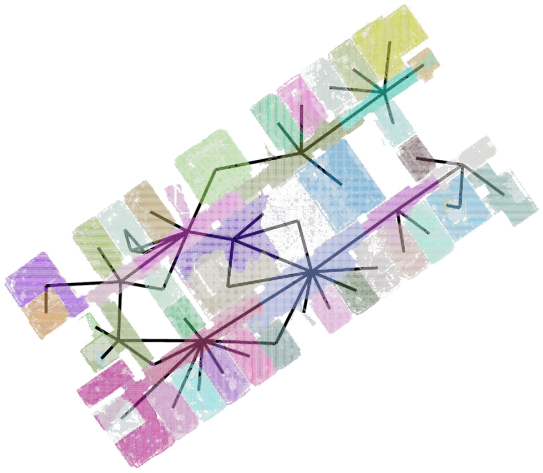

Figure 19: Second topometric map extracted from S3DIS area 1 dataset.

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.

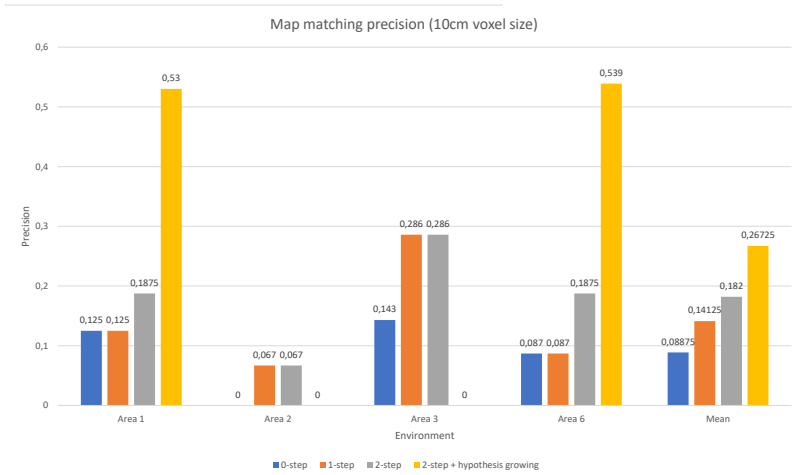

Figure 20: First topometric map extracted from S3DIS area 1 dataset.

Figure 21: First topometric map extracted from S3DIS area 1 dataset.

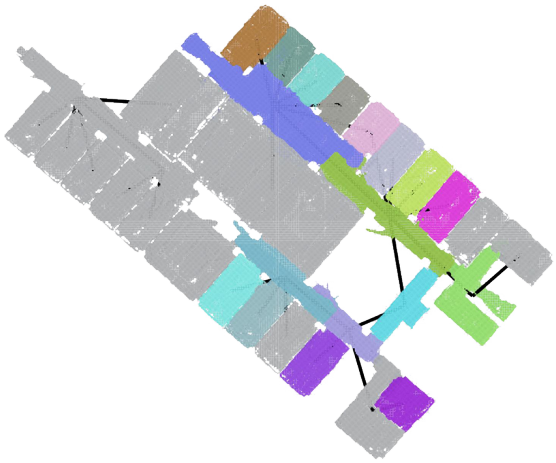

Figure 22: Second topometric map extracted from S3DIS area 1 dataset.

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

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

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

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