Map Merging of Heterogeneous Topometric Maps Using Attributed Node Embedding

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

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

24/01/2022

Contents

1 Introduction

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

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

In this thesis we propose to use both the hierarchical topological relationships of indoor environments, meaning the connectivity between distinctive places and their nesting relationships, and their metric characteristics, the geometry of the environment, to solve the heterogeneous map merging problem. We do this by extracting 3D hierarchical topological-metric maps from heterogeneous partial maps. We then use both the topological and metric characteristics of the partial maps to robustly identify overlapping areas that might appear differently due to being captured by heterogeneous agents. Our hypothesis is that the connectivity and hierarchy 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 hierarchical topological-metric map merging to indoor environments. 2) extracting 3D hierarchical topological-metric maps from heterogeneous partial maps. In the rest of this section we will give more detailed definitions for the concepts mentioned above and others that are commonly used in this report.

1.1 Definitions

In this section we will give a number of definitions for the regularly used concepts in this thesis.

./figures/raster/map_merging.png

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

Map A symbolic representation of an environment which contains information about its characteristics.

Map Representation The choice of characteristics of the environment that a map shows. Hybrid map representations are representations that shows a combination of multiple characteristics of the environment, e.g. a hybrid topological-metric map containing both the environment's large-scale structure and small-scale geometry.

Agent According to **?** "An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.". In the context of this thesis an agent specifically refers to any human or robot that is capable of perceiving its environment by means of a monocular camera, stereo camera or lidar and is capable of exploring their environment. We further define heterogeneous agents as having different sensing capabilities and being unable to communicate or position themselves relative to eachother.

Partial Maps A collection of maps without a common coordinate frame that each represent a part of the environment. We denote the set of all partials maps as $I = \{m_i\}_{i=1}^{n}$, where n is the number of partial maps. We define heterogeneous partial maps as partial maps captured by heterogeneous agents, thus having different scale, resolution, accuracy or precision.

Global Map A single, more complete map constructed by merging multiple partial maps. To do so, a coordinate transformation T must be applied to the partial maps to bring them into a common coordinate frame. We denote the global map constructed by merging a subset of partial maps $J \subset I$ using coordinate transformations $\{T_i\}_{i=1}^{|J|}$ as G_J, such that $G_J = \{T_i(m_i)\}_{i=1}^{|J|}$.

Map Merging The map merging problem can be stated as follows: given two partial maps $m_1 \in I, m_2 \in I$, find the coordinate transformation T that minimizes a dissimilarity function $\psi(m_1, T(m_2))$ (**?**). The goal of this is to "maximize the overlap of regions that appear in two or more partial maps" (**?**). Depending on the approach, the coordinate transformation, dissimilarity function or optimization method differ.

Metric Map Metric maps represent the geometry of an environment. They are usually derived from sensor range measurements, either directly or by using structure from motion or simultaneous localization and mapping algorithms. A common metric map representation is the occupancy grid (**?**). Another common metric map representation is the point cloud, which represents the surface of the environment as a collection of points. Both map representations are shown in figure **??**.

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 **??**) (**??**). 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 (**??**). We denote the topology of an environment as a graph G, where vertices V represent n distinctive places v_i and edges E represent the presence of m traversable paths between neighbouring pairs of places $\{v_j, v_k\}$, such that $G = (V, E)$, $V = \{v_i\}_{i=1}^{n}, E = \{\{v_j, v_k\}_i\}_{i=1}^{m}, v_j \in V, v_k \in V$.

./figures/raster/topo.png

Figure 2: Diagram showing the topological map of a building with four rooms connected by doors or stairs.

In the context of indoor mapping the graph is often embedded in 2D or 3D euclidean space as a spatial graph. Given the embedding $f : G \to R^n$, $\widetilde{G} := f(G)$, we denote $\widetilde{G}$ as the spatial graph of G (**?**). Each vertex in $\widetilde{G}$ has an associated 3D coordinate describing its position in the coordinate frame of the map. If the specific paths between vertices are known they can be associated with the edges in $\widetilde{G}$, otherwise an edge only represents a possible path between two points in space.

Point Cloud An unordered collection of points representing the geometry of an object or environment in 3D euclidean space, defined as $\mathcal{P} = \{p_i\}_{i=1}^{n}, p_i \in \mathbb{R}^3$, where n denotes the number of points (**?**). See figure **??** for an example of a point cloud.

Occupancy Grid Also known as a voxel grid, an occupancy grid is a "multi-dimensional (typically 2D or 3D) tesselation of space into cells, where each cell stores a probabilistic estimate of its state." (**?**). A 3D occupancy grid with a number of cells along each dimension, $O_{dim} \in \mathbb{Z}^3_{\geq 0}$ can be represented by a collection of positive integer cell indices that have a non-zero chance of being occupied $O = \{c_i\}_{i=1}^{n}$, $n \in \mathbb{R}$, $n \in [1, \prod_{i=1}^{3} O_{dim,i}], c \in \mathbb{Z}^3_{\geq 0}$. The probability of a cell being occupied is given by a cell's associated state variable $s(c) \in \mathbb{R}$,

$s(c) \in [0,1]$. Cells that are within the extent of the grid that are not stored have zero chance of being occupied, such that $\forall c \in \{c \notin O | c_i \in [0, O_{dim,i}]\} \Rightarrow s(c) = 0$. In the case of a binary occupancy grid a cell can either be occupied or not, such that $s(c) \in \{0,1\}$. In this case, explicitly storing a state variable is not necessary as only the cells in O are occupied, as given by $\forall c \in O \Rightarrow s(c) = 1$. See figure **??** for an example of an occupancy grid.

`./figures/raster/voxelization.png`

Figure 3: Example of an occupancy grid (right) derived from a point cloud (left).

Topological-Metric Map A hybrid map representation combining both the topological and metric characteristics of the environment. This map representation allows the end-user to use either topological or metric information depending on the needs of the situation, e.g. the topological layer can be used for large-scale navigation and abstract reasoning while the metric layer can be used for landmark detection or obstacle avoidance. In the context of this thesis a topological-metric map refers to a 3D representation of an environment containing both a metric occupancy grid map $\mathbb{M}_M$ representing its geometry and a spatial graph $\widetilde{\mathbb{M}_T} = (V, E)$ representing its topology. Each vertex $v \in V$ has an associated variable $m(v)$ representing a subset of the full metric map describing that place, such that $m(v) \subset \mathbb{M}_M$. See figure **??** for an example of a topological-metric map.

2 Map Extraction

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

2.1 Overview

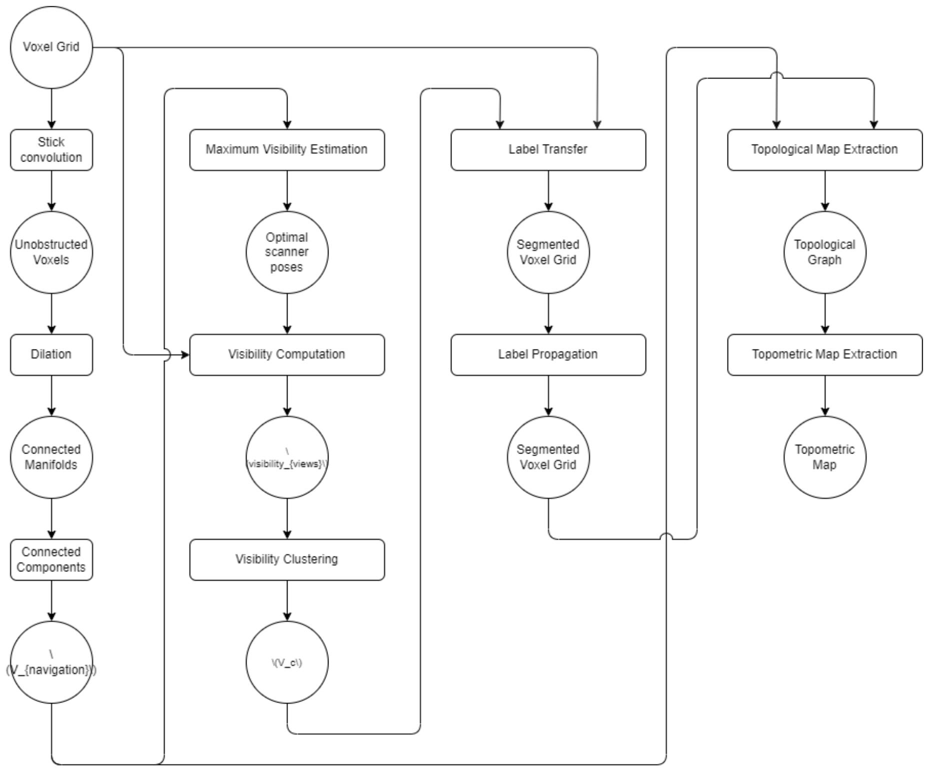
The goal of the map extraction step is to transform a partial voxel grid map $\mathcal{V}$ into a topometric map $\mathcal{T}$. It works by first finding a navigation graph $\mathcal{G}_{navigation}$ that connects a subset of the partial map $\mathcal{V}_{navigation} \in \mathcal{V}$, which represents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

2.2 Navigable volume

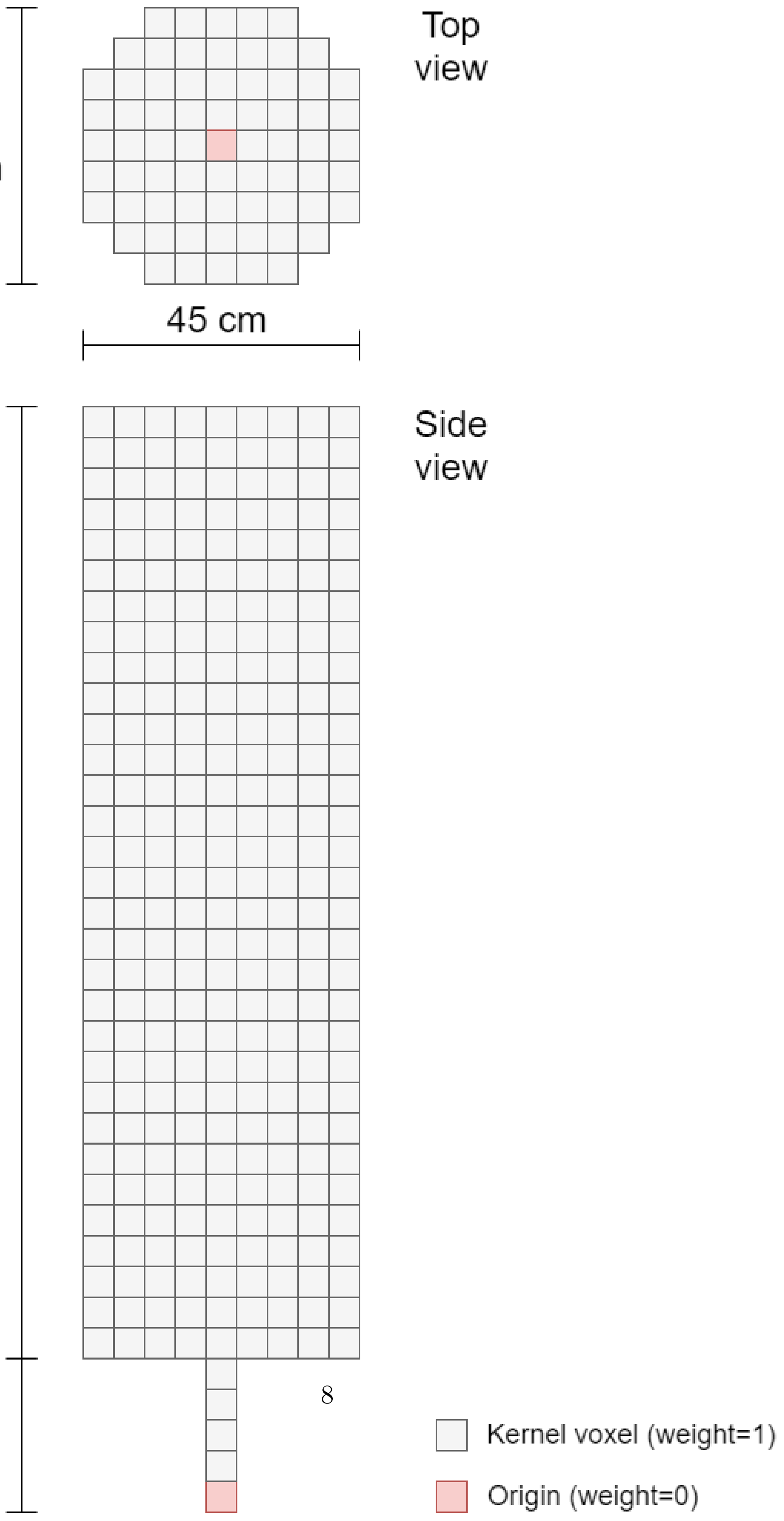
In our first step we extract a navigation graph $\mathcal{G}_{navigation}$. The navigation graph tells us how a theoretical agent in the map's environment would navigate through that environment. In practice, this means the areas of the floor, ramps and stairs that are at a sufficient distance from a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

2.2.1 Convolution

The first step uses voxel convolution with a stick-shaped kernel $\mathcal{K}_{stick}$ (shown in **??**) 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 results in voxels with an obstructed value of 0 when no other voxels are in the stick kernel. This indicates that these voxels have enough space around and above them to be used for navigation. We denote this convolution as $\mathcal{V}_{\mathcal{K}_{stick}} = c(\mathcal{V}, \mathcal{K}_{stick})$. We then filter out all non-zero voxels, such that $\mathcal{V}_{unobstructed} = \{\boldsymbol{v} \in \mathcal{V}_{\mathcal{K}_{stick}} \mid \mathcal{V}_{obstructed}(\boldsymbol{v}) = 0\}$.


Voxel Grid
Stick convolution
Unobstructed Voxels
Dilation
Connected Manifolds
Connected Components
\(V_{navigation}\)
Maximum Visibility Estimation
Optimal scanner poses
Visibility Computation
\(visibility_{views}\)
Visibility Clustering
\(V_c\)
Label Transfer
Segmented Voxel Grid
Label Propagation
Segmented Voxel Grid
Topological Map Extraction
Topological Graph
Topometric Map Extraction
Topometric Map

Stick kernel with edge length of 5cm



2.2.2 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 $\mathcal{V}$.

2.2.3 Connected components

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

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

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
```

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 intersection of the navigation graph's nodes and the whole voxel grid map as $\mathcal{V}_{navigation} = \mathcal{V}_{max} \cap \mathcal{V}$.

2.3 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.

2.3.1 Horizontal distance field

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

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 $HDF_{max} = \{\boldsymbol{v} \mid dist(\boldsymbol{v}) \geq max\{dist(\boldsymbol{v_r} \mid \boldsymbol{v_r} \in radius(\boldsymbol{v},\ r))\}\}$. Increasing the value of r reduces the number of local maxima and vice versa. All voxels in HDF_{max} lie within the geometry of the environment, which means the view of the environment is blocked by the surrounding voxels. To solve this, we take the centroids of the voxels in HDF_{max} and translate them upwards to a reasonable scanning height h, to estimate the positions with the optimal view of the map. We denote these positions as $views = \{\boldsymbol{v_c} + (0, h, 0) \mid \boldsymbol{v} \in HDF_{max}\}$. We use the following algorithm to compute $views$.

Algorithm 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
```

2.4 Visibility

The next step in the room segmentation algorithm is to compute the visibility from each position in $views$. We denote the all voxels that are visible from a given position as $visibility : \mathbb{R},\ \mathbb{Z}^{n\times 3} \mapsto \mathbb{Z}^{m\times 3},\ m \in \mathbb{R},\ n \geq m$. A target voxel is visible from a position if a ray cast from the position towards the centroid of the voxel does not intersect with any other voxel. To compute this we use the digital differential analyzer (DDA) algorithm to rasterize the ray onto the voxel grid in 3D. We then check if any of the voxels that the ray enters- except the target voxel- is occupied. If none are, the target voxel is visible from the point. We perform this raycasting operation from every position in $views$ towards every voxel within a radius $r_{visibility}$ of that position. Only taking into account voxels within a radius speeds up the visibility computation, and is justifiable based on the fact that real-world 3D scanners have limited range. We denote the set of visibilities from each point in views as $visibility_{views} = \{visibility(\boldsymbol{x}) \mid \boldsymbol{x} \in views\}$. The below pseudocode shows how the visibility computation works.

Algorithm 4 Visibility

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

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
  $\boldsymbol{p}_{current} = \boldsymbol{o}$
  $\boldsymbol{v_o} = (\boldsymbol{o} - \mathcal{V}_{min}) // \mathcal{V}_e$
  $\boldsymbol{v}_{current} = \boldsymbol{v_o}$
  $\boldsymbol{d} = (\boldsymbol{t} - \boldsymbol{o})$
  $heading = \boldsymbol{d} \odot abs(\boldsymbol{d})^{-1}$       ▷ Determine if ray points in positive or negative
  direction for every axis
  while $(\boldsymbol{v}_{current} \notin \mathcal{V} \vee \boldsymbol{v}_{current} = \boldsymbol{v_o}) \wedge \boldsymbol{p}_{current} \in [\mathcal{V}_{min}, \mathcal{V}_{max}]$ 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
```

2.5 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 $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$. Computing the Jaccard index for every combination of visibilities gives us a similarity matrix $S^{n \times n} \in [0,1]$. The similarity matrix is symmetric because $J(A,B) = J(B,A)$. Its diagonals are 1, as $J(A,A) = 1$. We can also consider $S^{n \times n}$ as an undirected weighted graph $\mathcal{G}_S$, where every node represents a visibility and the edges the Jaccard index of two visibilities. This means we can treat visibility clustering as a weighted graph clustering problem. To solve this problem we used the Markov Cluster (MCL) algorithm (CITE MCL), which has been shown by previous research to be state-of-the-art for visibility clustering.

The main parameter of the MCL algorithm is inflation. By varying this parameter between an approximate range of $[1.2, 2.5]$ we get different clustering results. We find the optimal value for inflation within this range by maximizing the clustering's modularity. This value indicates the difference between the fraction of edges within a given cluster and the expected number of edges for that cluster if edges are randomly distributed.

We denote the clustering of $visibility_{views}$ that results from the MCL algorithm as $\mathbf{C}_{visibility} = \{c_0, c_1, \ldots c_{n-1}, c_n\},\ n = |views|,\ c \in \mathbb{Z},\ n \geq c$, where nth element of $\mathbf{C}_{visibility}$ is the cluster that the nth 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 $room : \boldsymbol{v} \mapsto \mathbb{Z}$, such that $room(\boldsymbol{v}) = c,\ c \in \mathbf{C}_{visibility},\ \boldsymbol{v} \in \mathcal{V}$. This often results in noisy results, with small, disconnected islands of rooms surrounded by other rooms. Intuitively, this does not correspond to a reasonable room segmentation. To solve this, we apply a label propagation algorithm. This means that for every voxel we find the voxels within a neighbourhood as defined by a convolution kernel. We then assign to the voxel the most common label, in this case the room, of its neighbourhood, if that label is more common than the current label. We iteratively apply this step until the assigned labels stop changing. Depending on the size of the convolution kernel the results are smoothed and small islands are absorbed by the surrounding rooms.

Algorithm 6 Label propagation

```
Require:   Voxel grid $\mathcal{V}$
Require:   Labeling $label : \mathbb{Z}^{3+} \mapsto \mathbb{Z}$
Require:   Kernel $\mathcal{K}$
Ensure:   Propagated labeling $label_{propagated} : \mathbb{Z}^{3+} \mapsto \mathbb{Z}$
  while ( do$label \neq label_{propagated}$) ▷ Keep iterating until labels stop changing
      Reset propagated labeling
      for each ( do$\boldsymbol{v} \in \mathcal{V}$)
          $\boldsymbol{v}_{neighbourhood} = neighbours(\boldsymbol{v}, \mathcal{K})$
          $L_{neighbourhood} = \{label(\boldsymbol{v}_{nb}) \mid \boldsymbol{v}_{nb} \in \boldsymbol{v}_{neighbourhood}\}$
          $l_{max} = argmax_l \ |\{l \mid l \in L\}|$      ▷ Get most common label in neighbourhood
          if $|\{l \mid l \in L \wedge l = l_{max}\}| > |\{l \mid l \in L \wedge l = label(\boldsymbol{v})\}|$ then  ▷ If most common label more common than current label, set it as propagated label
              $label_{propagated}(\boldsymbol{v}) = l_{max}$
          else
              $label_{propagated}(\boldsymbol{v}) = label(\boldsymbol{v})$
          end if
      end for
      $label = label_{propagated}$      ▷ Use propagated labeling as input for next iteration
  end while
```

2.6 Topometric map extraction

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

3 Map Matching

The process of identifying overlapping areas between partial maps is called map matching. In the case of topometric map matching, this refers to identifying which nodes represent the same rooms between two partial maps. We denote our two partial topometric maps as $\mathcal{T}_A = (\mathcal{G}_A, \ \mathcal{V}_A)$ and $\mathcal{T}_B = (\mathcal{G}_B, \ \mathcal{V}_B)$. The

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
```

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

3.1 Geometrical Feature Embedding

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

3.1.1 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.

3.1.2 ShapeDNA

3.1.3 Deep Learning

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

PointNet

DGCNN

3.2 Attributed Node Embedding

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

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

SINE

MUSAE

FEATHER-N

3.3 Feature Matching

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