1 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.

1.1 Overview

1.2 Voxel representation

A voxel is the 3D equivalent of a pixel. A voxel represents a single cell in a bounded 3D volume divided into a regular grid, a voxel grid, and its associated properties. For example, a voxel may contain information about whether it is occupied and what its color is. We consider a voxel as a three-dimensional vector representing its coordinates along the x, y and z axes of the voxel grid.

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

Due to the regularly spaced nature of the voxel grid a voxel coordinate only consists of integer values. Voxel $\boldsymbol{v_{V_{min}}} = (0, 0, 0)$ represents the first cell along each of the voxel grid's axes and the minimum of the volume's bounds, voxel represents $(0, 1, 1)$ the first cell along the x and the second along the y and z axes, etc. We also restrict voxel coordinates to only be positive as negative coordinates would fall outside of the bounds of the volume. For the same reason, a voxel's coordinates can not be larger than that of the voxel representing the volume's maximum bounds $\boldsymbol{v_{V_{max}}} = (V_{max} - V_{min})//e_l$. We define a voxel grid as a set of occupied voxels $\mathcal{V} = \{v_i\}_{i=1}^{n},\ n \in [1, \prod \boldsymbol{v_{V_{max}}}]$ with an associated bounded volume $V_{\mathcal{V}}$ and edge length e_l.

1.2.1 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 coordinates into a single binary number, called a Morton code. The ordered vector of Morton codes gives the Morton order. We define the Morton order of a voxel grid as $M_{\mathcal{V}} = \{m_i\}_{i=1}^{|\mathcal{V}|},\ m_i < m_{i+1},\ m_i \in \mathbb{Z}^+$. We then divide the Morton order into buckets with width 8, such that each bucket contains at most 8 Morton codes, with a maximum difference of 8, in Morton order. Each non-empty bucket represents a parent node of at most 8 child nodes in the octree. By recursively performing this step until only one bucket remains, the root node, a sparse voxel octree is constructed.

1.3 Navigable volume

To extract the topology from the voxel grid map we first extract a navigable volume. The navigable volume tells us which voxels a theoretical agent in the environment would use to navigate through that environment. In practice, this means the areas of the floor and stairs that are at a sufficient distance from a wall and a ceiling. We compute the navigable volume using a three step algorithm.

1.3.1 Voxel convolution

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

The first step in the navigable volume extraction uses voxel convolution with a stick-shaped kernel $\mathcal{K}_{stick}$ based on the research by (GORTE REFERENCE). 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 possibly be navigable. 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 $\mathcal{V}_{unobstructed} = \{\boldsymbol{v} \in \mathcal{V}_{\mathcal{K}_{stick}} \mid \mathcal{V}_{obstructed}(\boldsymbol{v}) = 0\}$.

1.3.2 Dilation

The next step of the algorithm is to dilate the unobstructed voxels upwards by 20-25cm. This connects the unoccupied voxels separated by a small height differences into a connected volume. The result is a new voxel grid $\mathcal{V}_{dilated}$.

1.3.3 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 $\mathcal{G}_{\mathcal{V}} = (V,\ E)$ 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 $\mathcal{V}_{cc} = \{\mathcal{V}_{c,\ i}\}_{i=1}^{n}$. Finally, we extract the navigable volume $\mathcal{V}_{nav}$ by finding the connected component with the most voxels.

1.4 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 the navigable volume. The steps to achieve this are as follows.

1.4.1 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 distance of a voxel to its boundary as $dist : \mathbb{Z}^{3+} \mapsto \mathbb{Z}$.

In the next step we detect local maxima in the vo

1.4.2 Local maxima

1.5 Visibility

1.5.1 Voxel raycasting

1.6 Room segmentation

1.6.1 Visibility clustering

1.6.2 Similarity measure

1.6.3 Markov Clustering

1.6.4 Label propagation

1.7 Topometric map extraction

Pintore et al (2020) reviews the state-of-the art of 3D reconstruction of structured indoor environments. Ochmann et al (2019) uses visibility.