Autonomous Robots manuscript No.
(will be inserted by the editor)

Map Evaluation using Matched Topology Graphs

Sören Schwertfeger · Andreas Birk

Received: date / Accepted: date

Abstract Mapping is an important task for mobile robots. The assessment of the quality of maps in a simple, efficient and automated way is not trivial and an ongoing research topic. Here, a new approach for the evaluation of 2D grid maps is presented. This structure-based method makes use of a Topology Graph, i.e., a topological representation that includes abstracted local metric information. It is shown how the Topology Graph is constructed from a Voronoi Diagram that is pruned and simplified such that only high level topological information remains to concentrate on larger, topologically distinctive places. Several methods for computing the similarity of vertices in two Topology Graphs, i.e., for performing a place-recognition, are presented. Based on the similarities, it is shown how subgraph-isomorphisms can be efficiently computed and two Topology Graphs can be matched. The match between the graphs is then used to calculate a number of standard map evaluation attributes like coverage, global accuracy, relative accuracy, consistency, and brokenness. Experiments with robot generated maps are used to highlight the capabilities of the proposed approach and to evaluate the performance of the underlying algorithms.

Keywords Mobile robot · Performance metric · Map quality · Ground truth comparison · Topology · Place recognition · Simultaneous Localization and Mapping (SLAM)

Sören Schwertfeger
School of Information Science and Technology,
ShanghaiTech University, 200031 Shanghai, China
E-mail: soerensch@shanghaitech.edu.cn

Andreas Birk
Department of Electrical Engineering and Computer Science,
Jacobs University Bremen, 28759 Bremen, Germany
E-mail: a.birk@jacobs-university.de

1 Introduction

The development in the area of mobile robotics has been constantly gaining pace in the recent years. Many advanced capabilities of mobile robots require spatial information about the robot, the interaction partners, or of objects and places of interest. This information is usually stored in maps [31]. Even if a-priori information about the robot's environment is available, mapping is needed to deal with dynamic and changing environments.

Many robotic algorithms rely on good maps, one prominent example being autonomous navigation using path planning. Maps also assist an operator of a remotely tele-operated robot in locating the robot in the environment by providing information of features of interest like corners, hallways, rooms, objects, voids, landmarks etc. Those features are referenced in the map in a global coordinate system defined by the application. This frame of reference can be a geographic coordinate system of the earth, or a local one, defined by the application, e.g., the robot start pose or the pose of an operator station.

The quality of the mapping algorithms and the generated maps has to be ensured. In order to be able to specify and test the performance of mapping systems, the result - the maps - have to be analyzed and evaluated in a systematic, repeatable and reproducible way. Maps generated by mobile robots are abstractions of the real world, which always contain inaccuracies or errors, for example due to localization errors caused by bad odometry, inaccuracies in the sensor readings, inherent limitations of the used algorithms or dynamics in the environment. There has been great progress in mapping in the last two decades, particularly with respect to Simultaneous Localization and Mapping (SLAM) tech-

niques. But especially on extended missions or in unstructured environments, maps still often contain large errors.

The usefulness of a map not only depends on its quality but also on the application [18]. In some domains certain errors are negligible or not so important. That is why there is not one measurement for map quality. Different attributes of a map should be measured separately and weighed according to the needs of the application [17]. Those attributes can include:

- Coverage: How much area was traversed/visited.
- Resolution quality: To what level/detail are features visible.
- Global Accuracy: Correctness of positions of features in the global reference frame.
- Relative Accuracy: Correctness of feature positions after correcting (the initial error of) the map reference frame.
- Local Consistencies: Correctness of positions of different local groups of features relative to each other.
- Brokenness: How often is the map broken, i.e., how many partitions of it are misaligned with respect to each other by rotational offsets.

Note that these aspects tend to be influenced by several factors. The resolution quality for example not only depends on the actual size of the grid cells of the map, but it is also influenced by the quality of the localization and of the sensors. Pose errors between scans of the same object cause its features to blur or to completely vanish.

A number of approaches to the map evaluation problem have been proposed. Using the ground truth path of robots, these paths are compared in [36] and [15] to the pose estimations of SLAM algorithms. This principle can be applied in general to sensor datasets with accurately known ground truth poses of the places where the data was recorded. [29], [20] and [11] provide for example datasets of 3D sensor measurements and ground truth poses for benchmarking the performance of SLAM algorithms. The ground truth information has been obtained using a tracking system or by creating the data in a simulator, respectively. While these datasets are very useful contributions for benchmarking purposes, it is in general not feasible to obtain the actual robot locations for path-based evaluation, e.g., to test the performance of a complete system consisting of an arbitrary robot and sensor payload with an arbitrary SLAM algorithm in an arbitrary environment. In addition, it does not allow to compare maps that are generated with different control strategies, e.g., in exploration experiments, as the evaluation is bound to the specific path in which the data was gathered.

Most other approaches to map evaluation use ground truth data of the environment itself. These can for example be ground truth occupancy grid maps, which are particularly easy to obtain if simulations are used [27]. One can, for example, measure the alignment error of virtual scans in the ground truth map [16] or use image-based techniques. Image similarity methods [33] have their limitations due to the common errors in maps, because maps often have structural errors like noise, structures appearing more than once due to localization errors and the like. Nevertheless, certain attributes like the level of brokenness [6], can still be obtained.

In place-based map evaluation approaches features are quite often detected in the maps by treating them like images. The Harris corner detector, the Hough transform and the Scale Invariant Feature Transform (SIFT) are e.g. used in [35] while Speeded Up Robust Features (SURF) and a room detection method are used in [19]. Those approaches have in common, that they use the pose of the detected features to determine the map quality. They also require detailed ground truth representations of the environment to be usable. A combination of methods is used in the RoboCup Rescue Virtual Robots competition [4].

To ease place-based evaluation the use of artificial markers in the environment, called Fiducials, has been proposed [26,25]. In this place-based approach just the positions of those Fiducials has to be known to calculate the map attributes mentioned above. The main disadvantage of this approach to map evaluation is, that the mapped area as to be populated with those markers.

The topological approach on map evaluation presented in this paper has already been outlined by the authors in [23] and described in this thesis [22].

The rest of this article is structured as follows. Section 2 gives an overview and introduces the notations used in this article. Section 3 introduces the generation of the Topology Graph, which is mainly based on a Voronoi Diagram computation and simple pruning strategies, especially the concentration of the graph structure on junctions and dead-ends. Section 4 presents different methods to calculate the similarity of vertices from two graphs. This includes several substantially different strategies, especially the use of place-recognition based on local occupancy information, i.e., sensor-data, as well as the investigation of alternative methods based on topological information. Experiments are presented that benchmark the different algorithms for vertex similarity. In Section 5 an algorithm for matching two Topology Graphs is introduced. It is shown in Section 6 how a match can be used for map evaluation, i.e., how to compute the different map quality attributes from it, and

experimental results for map evaluation are presented. Section 7 concludes this article.

2 Overview and Notations

The input to our map evaluation are two dimensional grid maps, i.e., arrays of free or occupied cells [25]. We assume w.l.o.g. that any kind of "geo-referencing" information to project the maps to real-world metric coordinates, especially offsets and scalings, can be a priori extracted and stored as separate parameters. A map m can hence be formalized as a function $m : \mathbb{N} \times \mathbb{N} \to \{0, 1\}$, where 0 represents free and 1 occupied space. A grid cell at the coordinates (i, j) is denoted with $m[i, j]$ with $i, j \in \mathbb{N}$.

We use topological structures defined via a graph $G = (V, E)$ with a set of vertices $V = \{v_0, ..., v_k\}$ and a set of edges $E = \{e_0, ..., e_l\} \subseteq V \times V$. The idea is as usual with topological representations that vertices represent places and edges represent connections between them.

We also maintain metric information for the components of the graph G within the reference frame of the grid map m from which G is derived from. This is reflected by using a labeled graph. The locations of the vertices and of the connections between them are real valued. Though being formally two different functions, the two labeling functions for V and E are denoted with $L()$ for the sake of convenience:

$$L : V \to \mathbb{R}^2, L(v) = (x, y)$$

$$L : E \to (\mathbb{R} \times \mathbb{R})^k, L(e) = ((w_0), ..., (w_{k-1}))| \\ e = (v_1, v_2) \wedge w_0 = v_1 \wedge w_{k-1} = v_2$$

So, the label of a vertex v represents the exact location (x, y) of this place in the frame of map m. The vertices are as mentioned real-valued. This allows for example to place a vertex v that is derived from the computation of a Voronoi Diagram between the two grid cells $m[0, 0]$ and $m[1, 0]$ at the location $L(v) = (0.5, 0.0)$.

The label of an edge $e = (v_1, v_2)$ is a sequence of $\mathbb{R}^2$ map coordinates, i.e., an ordered list of k waypoints that form a path from v_1 to v_2. Note that the term "path" is used here as a matter of convenience; the "waypoints" are derived from the computation of the Voronoi Diagram and its subsequent processing as explained later on. They are not related in any way to the path of the robot that generated the map. The path $L(e)$ that is associated with each edge e is not required to be a single straight line - it is polyline between the waypoints. Given an edge $e = (v_1, v_2)$, the term *exit* of e from v_1 is used to refer to the start of the path $L(e)$ from $L(v_1)$, e.g., to refer to local metric information like the rotation-invariant relative angles of all exits from vertex v_1.

Appending an element to an ordered list is denoted in this article with a dot; we use for example following pseudo-code to extent the path associated with an edge e with the real-valued coordinates of vertex v as additional waypoint: $L(e) = L(e) \cdot L(v)$. As explained in detail later on, these paths - as well as the whole structure of the graph G - are derived from the Voronoi Diagram of the grid map m.

The typical input form of a map is a rectangular $m \times n$ array m_{in}. The area of interest of the map where the evaluation is applied to can nevertheless have an arbitrary shape; we use for example as explained later on an alpha-shape operator to exclude unmapped parts from m_{in}. We therefore denote the area of the locations that a map covers with $A_m \subset \mathbb{R}^2$. A *ray* is a special edge $e \in E$ that connects to a vertex $\hat{v}$ that is located out of the of the area A_m, i.e.,:

$$ray(e = (v, \hat{v})) = \texttt{true} \iff L(v) \in A_m \wedge L(\hat{v}) \notin A_m$$

A ray can for example be generated by the Voronoi Diagram computation when an open corridor ends at the boundary to unmapped territory.

Suppose $\|(x_1, y_1), (x_2, y_2)\|_2$ denotes the Euclidean distance between two locations $(x_1, y_1), (x_2, y_2) \in \mathbb{R}^2$. The length $len()$ of an edge $e = (v_1, v_2)$ is then defined as the Euclidean distance between the locations of its vertices, i.e.,:

$$len() : E \to \mathbb{R}, \quad len(e) = \|L(v_1), L(v_2)\|_2$$

Furthermore, the length $len(L(e))$ of a path $L(e) = (w_0, ..., w_{n-1})$ of k waypoints w_i is defined as:

$$len() : (\mathbb{R} \times \mathbb{R})^k \to \mathbb{R}, \quad len(L(e)) = \sum_{i=0}^{k-2} \|w_i, w_{i+1}\|_2$$

As mentioned before, a path associated with an edge is not necessarily a straight line and it hence holds that $len(L(e)) \geq len(e)$.

Given two maps m and m' - where m is typically a ground truth representation - the goal is to compute an assessment of the quality $Q(m', m)$ of m' relative to m. This is done using the topological structures, i.e., using two labeled graphs $G = (V, E)$ and $G' = (V', E')$ that are matched against each other. Note that the quality $Q(m', m)$ is usually not just a single value but that it

is typically a multivalued vector composed of several different attributes as motivated in the introduction.

A similarity function $sim() : V \times V' \rightarrow \mathbb{R}$ is used to find places that so to say look alike. This similarity is the basis for finding correspondences between vertices $v_k \in V$ and vertices $v_l \in V'$. (Assumed) correspondences are denoted with $v_k \rightleftharpoons v_l$. The correspondences are one to one mappings and they can then be used to identify matches between the evaluated map and the ground truth. Up to here, this is conceptually similar to previous work on map evaluation using place recognition like [35,19,26,25], which mainly differ in the computation of $sim()$ - using different forms of natural landmarks or even artificial markers.

Here, the topological structure, i.e., connections between places, are also taken into account. This has three major advantages:

1. Considering every grid cell neighborhood as a potentially recognizable place in a map m is computationally expensive. Also, many cell neighborhoods are not particularly distinctive. The Voronoi Diagram that we use to derive the graph G from m provides inherently *few*, typically quite *distinctive* candidate locations for place recognition, namely the junctions of paths in the Voronoi Diagram.
2. The vertex similarity $sim()$ can exploit local neighborhood information, e.g., if $sim(v_k, v'_l)$ is very high and we hence assume that $v_k \rightleftharpoons v'_l$ then there should also be high similarities between some neighbors of v_k with some neighbors of v'_l. Using these relations makes the computation of $sim()$ much more robust and leads to proper determination of correspondences of places even between a very noisy, distorted map and a clean ground truth.
3. Given initial (possible) correspondences, the labeled edges can be used to compute subgraph isomorphisms between G and G'. This allows the identification of corresponding partitions of m and m', e.g., to determine high-level map evaluation criteria like brokenness.

Based on these concepts we explore different options for the computation of $sim()$ and for the computation of the subgraph isomorphisms. But first, the way the graph G is generated from m is presented in the following section.

3 Computation of a Topology Graph

3.1 Overview

Our extraction of topologies from robot generated maps is related to the generation of skeletons in computer vision [28,3], especially in the context of skeleton matching in applications like object recognition [32,2]. But there are several aspects that are quite specific to map evaluation. First, skeleton matching for object recognition is typically applied to the full silhouette of an object; a map that is to be evaluated is in contrast typically not a full representation of the ground truth, i.e., this would correspond to a situation with occlusions and partial views in the case of object recognition. Second, there is typically a high amount of noise in the maps whereas object silhouettes are typically assumed to be well segmented. Third, the errors in mapping can lead to non-rigid transformations, e.g., in the case of a broken region.

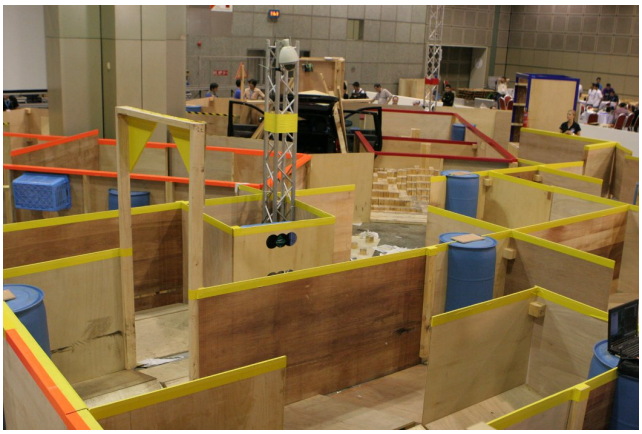


Fig. 1 Photo of the RoboCupRescue 2010 arena. Variations of this environment are represented in Figures 2 and 12 and used in the experiments in Section 6.3.

While the above aspects make map evaluation harder than skeleton based object recognition, we can also profit from special aspects of dealing with maps. For example, we assume the presence of corridors and junctions, i.e., of specific environment structures that can be exploited for simplification of the skeleton and for matching. Furthermore, the Topology Graph maintains abstracted metric information including especially local metric information like the relative angles in which edges - or more precisely, the related metric paths - leave a vertex. This abstracted metric information can be used for place-recognition.

We use as main basis a very standard approach for skeletonization, namely a Voronoi Diagram [34], computed with the centers of the obstacle cells of the 2D grid map as sites. The Topology Graph G is then derived through simplifications and pruning from the Voronoi Diagram VD. As an example of the desired output, Figure 2 shows a map from the RoboCup Rescue Competition in Singapore 2010 and its Topology Graph; Figure

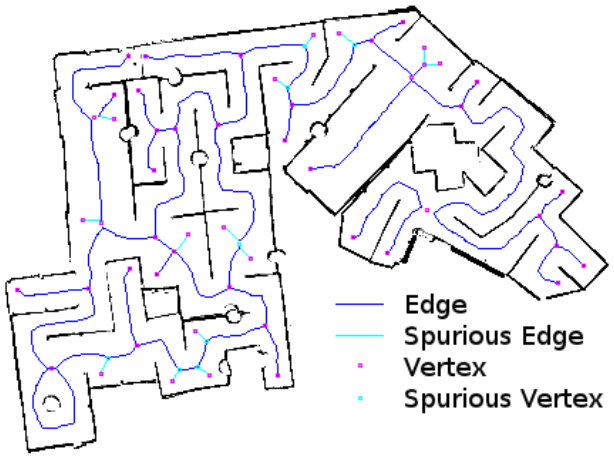


Fig. 2 A Topology Graph in a map from the RoboCup Rescue 2010 maze.

1 shows a photo of the maze wherein the map was generated.

In the following, a short overview of the steps for generating a Topology Graph is given first; a more detailed description of the steps with pseudo-code follows in sections 3.2 to 3.7.

3.1.1 Initial Voronoi Diagram Calculation with CGAL

The input to the Voronoi Diagram computation is a robot generated 2D grid map that is converted into a set M of occupied cells, with i and j being the row and column of the grid cell:

$$M = \{\ (x, y) \mid m[i, j] = 1, (i, j) \in \mathbb{N} \times \mathbb{N}\}$$

The Voronoi Diagram VD^0 is created using the standard implementation from the Computational Geometry Algorithms Library (CGAL). The set of row and column coordinates of the occupied cells are used as the input points (see Section 3.2 for more details):

$$M \xrightarrow{\text{Voronoi}} \mathrm{VD}^0 = \left(\mathrm{V}^{\mathrm{VD}^0}, \mathrm{E}^{\mathrm{VD}^0}\right)$$

During the computation of VD^0, CGAL also determines for each edge E^{VD^0} the geometric distance $d^{min}(e) \in \mathbb{R}$ to the closest occupied cell from M. This information will be used in the next step.

3.1.2 Filter with Obstacle Distance

Now a new graph VD^1 is created, which filters out all edges from VD^0 whose distance to the next occupied cell is smaller than a threshold value t_d (see Section 3.3 for more details).

$$\mathrm{VD}^0 \xrightarrow{\text{obst. distance}} \mathrm{VD}^1 = \left(V^{\mathrm{VD}^1}, E^{\mathrm{VD}^1}\right)$$

$$V^{\mathrm{VD}^1} = V^{\mathrm{VD}^0}$$

$$E^{\mathrm{VD}^1} = \left\{e \in E^{\mathrm{VD}^0} \mid d^{min}(e) > t_d\right\}$$

The main motivation is to keep only edges that belong to passable structures like corridors or doorways. The threshold is accordingly just set to a value that corresponds to 30 cm in the real world.

3.1.3 Topology Generation with only Dead Ends and Junctions by Edge Skipping

Next, the first level Topology Graph G^0 is generated based on the strategy to only keep dead-ends and junctions as main places in the graph. First, all vertices with a degree of zero or two are filtered out, i.e., only dead end vertices and junction vertices are left in the graph. Second, edges connected to filtered out vertices are merged; this includes the merging of the paths that are associated with them. This is done by starting a (1D) wavefront propagation algorithm along all edges of every vertex from V^0.

$$\mathrm{VD}^1 \xrightarrow{\text{edge skipping}} G^0 = \left(V^0, E^0\right)$$

$$V^0 = \{v \in V^{\mathrm{VD}^1} \mid deg\,(v) = 1 \vee deg\,(v) > 2 \vee (deg\,(v) == 2 \wedge \exists e \in v \mid ray(e))\}$$

$$E^0 = \{e = (v_1, v_2) | v_1, v_2 \in V^0;\ e \in E^{\mathrm{VD}^1} \vee \forall \tilde{v}_i \in V^{\mathrm{VD}^1} \setminus V^0 : (v_1, \tilde{v}_1), (\tilde{v}_i, \tilde{v}_{i+1}), (\tilde{v}_{i+1}, v_2) \in E^{\mathrm{VD}^1}\}$$

3.1.4 Boundary Computation and Cleaning

The next step cleans the graph from superfluous data that is beyond the boundaries of the actual map. As mentioned already in Section 2, the input map is typically a rectangular array which includes unmapped areas at its outer parts. So a boundary for the map has to be found to remove unknown areas - unless they are explicitly represented in some form. To accommodate to the sometimes complex shapes of the maps, the alpha shape algorithm is used for this purpose (see Section

3.5 for more details). Edges where both vertices are outside of the alpha-shape are removed together with their vertices, while edges with one vertex inside the alpha-shape and one vertex outside of the alpha-shape are becoming rays.

Furthermore, dead end vertices and edges that are shorter than a certain threshold are removed , i.e., only proper dead-ends that are within the boundaries of the actual map are kept.

In addition, only the biggest connected graph is kept, or more precisely, the connected graph with the largest sum of the lengths of the paths of its edges is kept. All vertices and edges not belonging to the biggest connected graph are filtered out . The reason for this is simple, namely, the graph matching can only be applied to a properly connected graph. There is of course the option to apply the graph matching - and hence map evaluation - to all connected components separately. Note that it is rare that there are unconnected components. Normally, the robot that has generated the map needed free passages between all components to map them. Nevertheless, the effects of noise and structural errors can distort maps such that they appear to consist of disconnected components - the quality is then computed per default for the largest connected part.

$$G^0 \xrightarrow{\text{cleaning}} G^1 = \left(V^1, E^1\right)$$

$$G^0 \xrightarrow{\alpha \text{ shape}} \ldots \xrightarrow{\text{dead end filter}} \ldots \xrightarrow{\text{max conn. graph}} G^1$$

3.1.5 Final Pruning by Vertex Merging

In a simplistic environment with clean straight walls and junctions, we would be done now - each junction and dead end would be represented by a single vertex each. But in reality there is furniture, sensor noise, possibly even dynamic objects, etc. There can be hence be multiple vertices generated by the Voronoi Diagram within one physical junction or room in the environment - in a more or less arbitrary way due to effects of noise and dynamics. This is a problem if the same place gives rise to a (significantly) different number of vertices in the ground truth versus the evaluated map.

But fortunately, a simple heuristic can be used to avoid this effect, namely by merging vertices that are close to each other, i.e., within a single place, into a single vertex. The final step hence prunes short edges from E^1, i.e., there is a threshold t_{Vjoin} and the vertices connected to edges shorter than t_{Vjoin} are merged. Those short edges are removed while the edges that were connected to the original vertices are now connected to the new joined vertex.

$$G^1 \xrightarrow{\text{merging}} G^2 = \left(V^2, E^2\right)$$

3.2 Computation of the Voronoi Diagram

As mentioned before, the different steps to generate the Topology Graph are now explained in a bit more detail in the following subsections, starting here with the computation of the Voronoi Diagram.

The Generalized Voronoi Diagram (GDV) (also called Voronoi decomposition, the Voronoi tessellation, or the Dirichlet tessellation) is a geometric structure, which is widely used in many application areas [14,1]. It is a partition of the space into cells. The cells enclose a site, and in our application the site is the center of an occupied cell from the grid map. The Voronoi cell associated with a site is the set of all points whose distance to the site is not greater than their distance to any other site. In this work the interest is not so much in the Voronoi cells but in the graph VD^0 that is defined by the boundary of the cells. The Computational Geometry Algorithms Library (CGAL) [13] is used in our implementation for the computation of the Voronoi Diagram.

The Voronoi Diagram only serves as basis for further simplifications and pruning as motivated before. The according steps are explained in more detail in the following and illustrated using an example map (see Figure 7 for a big version together with the resulting Topology Graph), which based on the *random maze* of the Response Robot Evaluation Exercise 2010 [30]. This map has 4398 occupied points and a resolution of 5 cm per pixel.

3.3 Filtering Edges with Short Distances to Obstacles

As the Voronoi computation uses the 2D point set of the centers of the occupied grid cells as input, there are, among others, edges going between the centers of adjacent cells. The Voronoi graph VD^0 hence has edges going through directly adjacent obstacles, i.e., those edges go for example through walls (see Figure 3) as an unavoidable side-effect of the 2D point input. Fortunately, these edges can be filtered out in a trivial manner based on the minimum distance to the nearest occupied cell.

While these side-effects could be avoided by more complex pre-processing, this filtering step for edges close to obstacles, i.e., occupied cells, is needed anyway. After all, we are interested in edges representing traversable

areas, i.e., the goal is a topological graph representing the hallways, doorways and rooms. Therefore, Algorithm 1 removes all edges with a distance of less than t_d to the closest obstacle (Figure 4). This can be done very efficiently as for every CGAL half edge the face of the dual Delaunay representation stores the location of the point, i.e., of the occupied cell, closest to the edge.

Implementing this edge removal is hence quite simple. Every half edge in CGAL has information about the face it belongs to. Each face has exactly one site, which is the obstacle cell from the grid map. And by definition of the Voronoi Diagram, this is also the closest site to this edge.

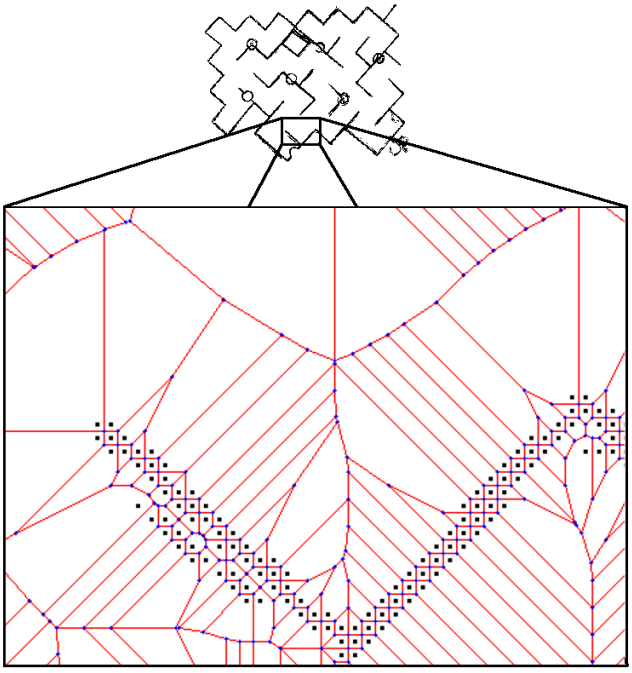


Fig. 3 A zoomed in view of a part of an unfiltered Voronoi Diagram. The centers of the occupied cells are shown as small squares.

Algorithm 1 Filtering edges close to obstacles.

```
for all e ∈ E^VD0 do
  if d^min(e) > t_d ∨ ray(e) then
    e ∈ E^VD1
  end if
end for
```

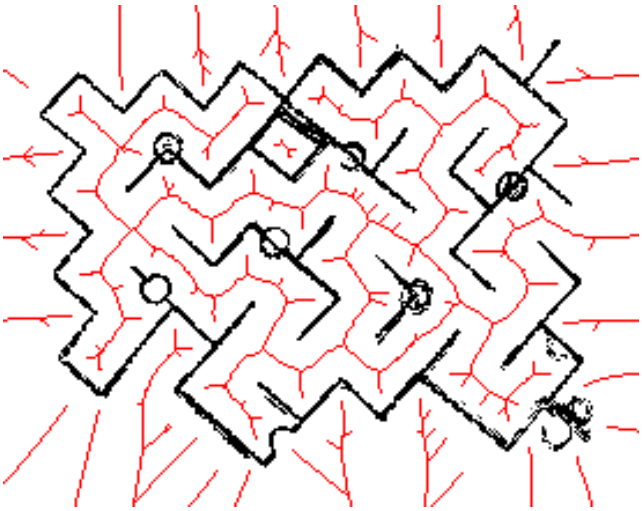
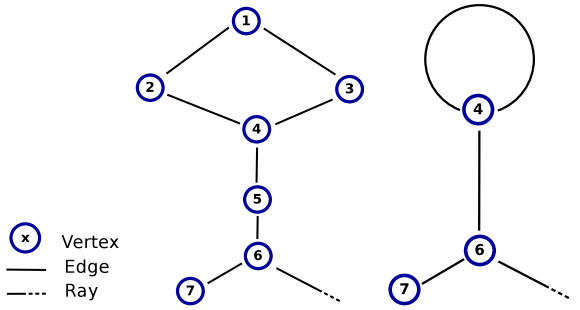


Fig. 4 The filtered Voronoi Diagram where edges with a minimum distance to occupied cells of less than 30cm are removed to concentrate on edges that are traversable. This reduces the number of CGAL half edges from 10,280 for the unfiltered graph to 2,352.

3.4 Edge Skipping

The main idea of this step is to keep only dead-ends and junctions. Therefore, unconnected vertices with degree 0 are removed and edges leading to vertices with degree 2 are merged. Figure 5 illustrates this.



(a) The original graph. (b) After edge skipping.

Fig. 5 The Edge Skipping algorithm merges edges leading to vertices with degree 2, i.e., only junctions and dead-ends are kept.

The algorithm works as shown in Algorithm 2: First, for all vertices, the number of edges connected to this vertex is counted. Then, the algorithm iterates over the vertices. All vertices that so to say just lay on a path, i.e., that have degree 2, are skipped. More precisely, these are the vertices that have two edges connected and where none of those edges is a ray. All other vertices are added to the new graph. In the example in Figure

5, we can see that the vertices 4, 6 and 7 are included in the new graph. Vertices 4 and 6 have 3 edges connected and vertex 7 just one. Even if there would be no vertex 7, vertex 4 would still be added to the new graph since one of its then two edges are rays. Vertices 1, 2, 3 and 5 are filtered out because they have exactly two edges.

Algorithm 2 Skipping vertices.

$V^0 = \{\}$
for all $v^{\mathrm{VD}^1} \in V^{\mathrm{VD}^1}$ **do**
  if $(deg(v^{\mathrm{VD}^1}) == 2)$ **then**
    if $\neg\exists e^{\mathrm{VD}^1}_{ray} : v^{\mathrm{VD}^1} \in e^{\mathrm{VD}^1}_{ray} \wedge e^{\mathrm{VD}^1}_{ray} \in E^{\mathrm{VD}^1} \wedge ray(e^{\mathrm{VD}^1}_{ray})$ **then**
      continue
    end if
  end if
  $V^0 = V^0 \cup \{v^{\mathrm{VD}^1}\}$
end for

Now we iterate through all vertices V^0 in the new graph and apply the edge skipping algorithm (see Algorithm 3) to all edges connected to the vertices. The algorithm is following the edges until it reaches a vertex with rays or a vertex with other than two edges. The vertex where it is stopping is then saved as the goal vertex for the new edge.

In the example, starting at vertex v_4, edge $e_{\{2,4\}}$ is subjected to the edge skipping algorithm. Edge $e_{\{2,4\}}$ is leading to v_2. Vertex v_2 has two edges, so it is skipped. The next edge is $e_{\{1,2\}}$. Since v_1 also has two edges, the next edge selected is $e_{\{1,3\}}$ and then $e_{\{3,4\}}$. v_4 has three edges, so the loop is broken and a new edge between the start vertex of the algorithm (v_4) to the last vertex (also v_4) is created. Now the edge skipping algorithm is applied to the other two edges leaving from v_4, too.

In the implementation several abstracted metric properties of the edges and vertices are calculated at the same time. The distance from vertex to vertex and the minimum distance to an obstacle are calculated in the Skip Edges Algorithm. Afterwards the distance for all vertices to their closest obstacle is gathered (using the distances from the neighboring edges) and the dead end vertices are marked. Figure 4 shows the color coded graphs. Please note that the graphs are actually much bigger than shown here, since they continue beyond the boundaries of the images.

3.5 Alpha Shape Boundary and Cleaning

In the context of map evaluation, only the parts of the graph the represent the actual map area are of interest. Since maps typically do not have a simple outer shape, bounding boxes are not an ideal option. So an Alpha

Algorithm 3 Skipping edges along one path.

Input: VD^1, start edge $e^{\mathrm{VD}^1}_{start} \in E^{\mathrm{VD}^1}$
Input: start vertex $v^{\mathrm{VD}^1}_{start} \in e^{\mathrm{VD}^1}_{start} : v^{\mathrm{VD}^1}_{start} \in V^0$
$v_{curr} = v^{\mathrm{VD}^1}_{start}$ Initialize current vertex
$e_{curr} = e^{\mathrm{VD}^1}_{start}$ Initialize the current edge
$l_{sum} = 0$ Initialize the sum of the lengths
$e_{new} = \{\}$ Initilize the new edge
while TRUE **do**
  $l_{sum} = l_{sum} + len(e_{curr})$ Add up the lengths
  $L(e_{new}) = L(e_{new}) \cdot [L(v_{curr})]$ Add the vertex coordinate to the path label
  $v_{next} \in e_{curr} \mid v_{next} \neq v_{curr}$ Set the next vertex
  if $deg(v_{next}) \neq 2$ **then**
    break
  end if
  The next vertex only has two edges - continue and select the next edge:
  $e_{next} \in E^{\mathrm{VD}^1} \mid v_{next} \in e_{next} \wedge e_{next} \neq e_{curr}$
  if $ray(e_{next})$ **then**
    break If this edge is a ray the loop is also broken
  end if
  $e_{curr} = e_{next}$
  $v_{curr} = v_{next}$
end while
The algorithm is finished - we have to add the new edge:
$L(e_{new}) = L(e_{new}) \cdot [L(v_{next})]$ Finalize the path label
$e_{new} = \{v_{start}, v_{next}\}$ Fill the new edge
$len(e_{new}) = l_{sum}$ Label the edge with the length
$E^0 = E^0 \cup \{e_{new}\}$ Add the edge to the graph

Shape [9] is used to define the outer boundary of the map. The biggest polygon generated by the alpha shape algorithm on the grid map is then used to define the outer boundary. All vertices outside this polygon are removed. Edges that have one vertex inside and the other outside are turned into rays. Figure 6 shows the alpha polygon and the filtered graph, now including many rays.

The Voronoi graphs can still be very detailed due to complexity in the environment, e.g., furniture, as well as due to sensing artifacts like noise or dynamics in the environment. This often leads to short dead end edges that for example point towards the walls. Hence all dead end edges that are shorter than a threshold are filtered out. After the removal of dead ends there might be (again) vertices with exactly two neighboring vertices. Those are removed and the edges are joined accordingly.

In addition, dead end edges that are shorter than a slightly larger threshold and their related vertices are marked as *spurious*. They are kept as potentially useful information in the graph while the spurious label indicates them as a possible sensing artifact that should be not fully trusted in topological place recognition. The term *major* vertex is used to explicitly denote vertices that are not spurious, i.e., that are considered to provide trustworthy topological information.

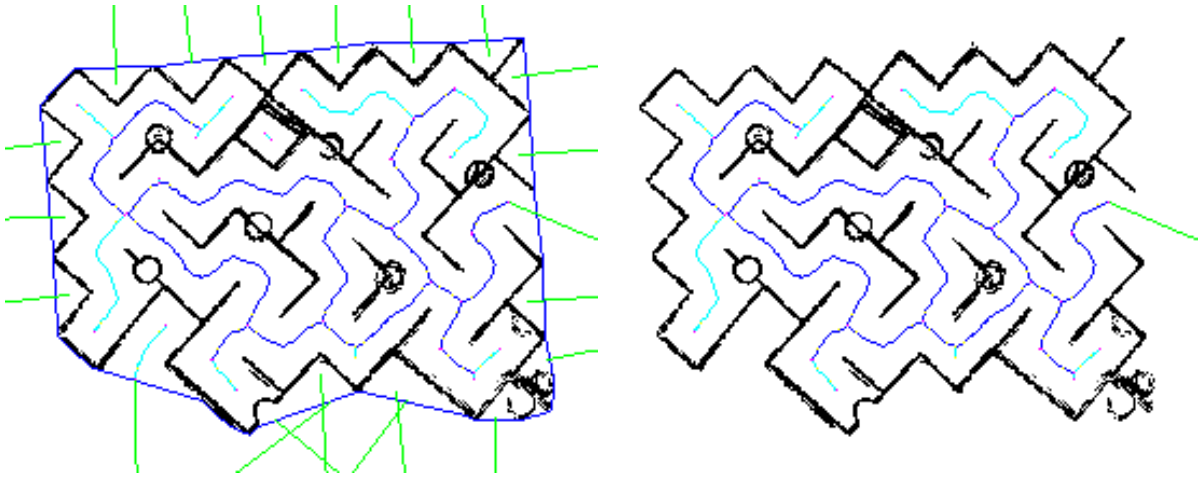


Fig. 6 A topology graph (left) after edge skipping, cleaning (filtered for short dead ends) and the application of the alpha-shape boundary (in blue) around the underlying grid map. The graph after keeping only the biggest connected graph is shown on the right. Ray edges are colored in green. An alpha value of 2500 (2.5 m) was used, resulting in an alpha polygon consisting of 133 segments.

Only a connected graph can be later-on used for the map evaluation. Thus all edges and vertices not part of the biggest connected graph are removed. The size attribute used to find the biggest graph is the sum of the length of all edges of a connected sub-graph.

3.6 Pruning by Vertex Merging

The final Topology Graph G^2 is generated by joining together vertices in close proximity to each other. The new vertex gets the average position of the joined vertices. The edges coming from other vertices to the joined vertices are connected to the new one. The related pseudo-code is shown in Algorithm 4.

Algorithm 4 Vertex Merging.

for all $e^{\mathrm{VD}^1} \in E^{\mathrm{VD}^1}$ **do**
  if $(len(e^{\mathrm{VD}^1}) < \sigma_{joinMax})$ **then**
    $v_1, v_2 \in e^{\mathrm{VD}^1}$
    $E^{\mathrm{VD}^1} = E^{\mathrm{VD}^1} \cap \{e^{\mathrm{VD}^1}\}$
    $V^{\mathrm{VD}^1} = V^{\mathrm{VD}^1} \cap \{v_1, v_2\}$
    $V^{\mathrm{VD}^1} = V^{\mathrm{VD}^1} \cup \{v_{new}\}$
    Set the label in v_{new} to the average position of v_1 and v_2
    for all $e \in E^{\mathrm{VD}^1}$ **do**
      if $\exists v : v \in e \wedge (v == v_1 \vee v == v_2)$ **then**
        Replace v with v_{new}
        $L(e) = L(e) \cdot [L(v_{new})]$
      end if
    end for
  end if
end for

As explained before, the motivation for this step is to merge vertices that are close to each other, i.e., that are likely within a single physical place, into a single vertex to represent that place. An example for a joined vertex can be seen in Figure 12: Vertex 18 of the left ground truth map was merged from two vertices. Figure 7 show the final graph for the example map used earlier.

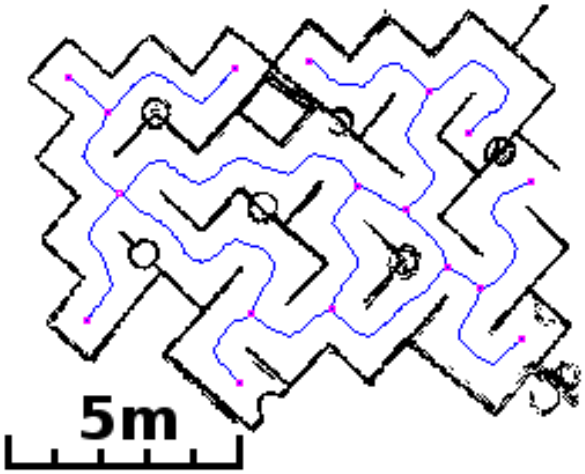


Fig. 7 The final Topology Graph with 17 vertices and 36 half edges.

The information that a vertex $\tilde{v}$ is generated by merging k vertices v_i is maintained in the topology graph by labeling $\tilde{v}$ as a *joined* vertex. The relative spatial locations of the vertices v_i with respect to the location $L(\tilde{v})$ of $\tilde{v}$ are also kept. They can be used as a local topological feature related to $\tilde{v}$ when looking for correspondences between vertices across maps.

3.7 Ordering of the Edges of a Vertex

To facilitate the subsequent calculations of topological vertex similarities, the sequence in which the edges connect to each vertex $v \in V^{G^2}$ of graph G^2 is determined, i.e., a deterministic spatial ordering based on the angle in which the associated paths are connected is computed for the edges of each vertex.

The goal of this step is hence to sort the edges connected to a vertex according to their incidence angle, such that the left- and right-hand neighbors of edges can be retrieved easily. This is not completely trivial, because the location of a goal vertex of the edge cannot be used - it might be somewhere completely different after a long and winding path. So a part of the path $L(e)$ associated to edge $e = (v, \tilde{v})$ close to the originating vertex v is used. Since this path might be curved, a number of points are sampled on the path and their angles are averaged. The edges are then put into a ring buffer, sorted by their global angle. But only the angle differences are stored, thus making this information rotation invariant.

4 Calculating the Similarity of two Vertices

In order to compare the Topology Graph of a ground truth map m with the one from a robot-generated map m', the graphs G_m and $G_{m'}$ have to be matched against each other. The first step for matching the graphs is to calculate the pair-wise similarities between the vertices from the two graphs. The positions of the vertices in the map are not used for this, because they - like any global metric information - can be severely wrong due to errors in the map. Nevertheless, some local metric information - like the relative orientations of the exits, i.e., of the paths leaving a vertex - can be used for establishing correspondences. Furthermore, purely topological properties - like the degree of a vertex - as well as appearance-based place recognition methods using local sensor data can be used for this purpose.

The Vertex Similarities that are presented in this section are always normalized such that their value is between 0 and 1 - where 1 corresponds to a perfect match and 0 corresponds to a maximum difference. As we will see in the following section, parts of the similarity score are often based on the difference $\delta()$ of the values of an attribute that is maintained in the Topology Graph for each vertex, e.g., on the difference of the respective distances to the nearest obstacle. To normalize scores that are based on differences of attribute values, we use an offset $off \in \mathbb{R}$ and a clipping value $max \in \mathbb{R}$ as follows:

$$\Delta(\delta(), off, max) = \begin{cases} 0 & \text{if } \delta() < off; \\ 1 & \text{if } \delta() > max; \\ \frac{\delta() - off}{max - off} \end{cases}$$

So, small differences below off are considered to be still perfect matches of the attributes while anything larger than max is considered to indicate no similarity at all. A side-effect of the use of differences of attributes is that a 0 (the absence of a difference) indicates the highest possible similarity and a 1 (substantial difference) indicates no similarity at all. As a matter of convenience, we hence define for a similarity score $s()$ its related dissimilarity score $\bar{s}()$ as $\bar{s}() = 1 - s()$ and vice versa.

The vertex similarity will guide the search for matching the graphs. It is thus desirable to use an algorithm which gives very good results for matching pairs while having a low complexity. Two substantially different approaches are investigated in this article, namely calculating similarities just using either

1. the information stored in the graph, i.e., topological methods (including the use of the abstracted metric data), or
2. the occupancy information of the local area around the vertices from the original maps, i.e., methods of sensor-based place recognition.

For each approach different algorithms and options are presented and tested. As shown later on, it turns out that the sensor-based methods deliver very good results. But methods just using information from the graphs perform very similar in terms of correspondence accuracies - while they are at least three orders of magnitude faster.

Once the similarities between all vertices are known and hence initial correspondence assumptions are established, one can boost the accuracy of the matching by checking the similarity values of neighboring vertices. If a correspondence between two vertices is correct, their neighbors should also correspond, i.e., they should all also have high similarities. This idea can be propagated to the neighbors of the neighbors, and so on. High, respectively low consistency in the similarities of the neighbors of similar vertices can hence be used to increase, respectively decrease the previously computed similarities. This leads to a strengthening of correspondence assumptions, respectively to their weakening up to a level where the correspondence assumption does not hold anymore. The related strategy for consistency checking is hence called a *Propagation Round* as it leads to a strengthening of consistent correspondence while it breaks up inconsistent correspondences - for which

then new assumptions based on the updated similarities must be found. This strategy can be applied to boost the robustness of all similarity algorithms; a detailed description is presented in Section 4.1.

In Section 4.2 the *Local Vertex Similarity* is introduced as a first method for computing a vertex similarity. It just uses topological information like the vertex degree as well as simple local metric information that is maintained in the Topology Graph like the relative orientation of paths leaving a vertex. The *Enhanced Vertex Similarity* presented in Section 4.3 also just uses information from the Topology Graph. It is based on the Local Vertex Similarity in combination with the Propagation Round strategy, which is supplemented by additional information from the graph.

Appearance-based similarity methods that use local occupancy information for place recognition are presented in Section 4.4; more precisely, ICP (Section 4.4.1), respectively iFMI (Section 4.4.2) are used as registration methods on local grid map data to check for correspondences between the related vertices.

In Section 4.5 the different algorithms for vertex similarity are experimentally compared.

4.1 Propagation Rounds

Once similarity values between all permutations of pairs of vertices from two maps have been calculated, it is possible to increase the robustness of correspondences by taking the similarity values from the neighboring vertices of the Topology Graphs into account. Thus vertices which have neighbors that also match nicely get higher similarity values, i.e., the correspondence assumption is hence strengthened, while the similarity value of vertices with more incompatible neighbors is decreased, i.e., the correspondence assumption is weakened and possibly even dropped.

A single update of the similarities of all assumed correspondences based on the consistency of the similarities of their neighbors is called a *Propagation Round*. Consecutive runs of the Propagation Round algorithm, each taking the vertex similarities from the previous round as input, lead to increased stability in the correspondences. In addition, further propagation rounds take more indirect neighbors into account.

Though the concept is simple, the actual implementation is not completely trivial. This is because the matching of vertices (v_x and v'_y) between the two graphs (G and G') is not (yet) determined. Suppose the set of the direct neighbors of a vertex v_x from G is $n(v_x)$ and the i-th neighbor is $n(v_x)_i$. Let us assume that v_x and v'_y are vertices from G and G' for which the consistency check has not been completed yet. The correspondence between the two sets of neighbors is then still an open issue. The main idea of the consistency check is to try all possibilities, to then assign a combined similarity to each of the permutations of possible neighbor correspondences, and to take the best combined similarity to boost correct correspondence assumptions. Two variants of this Propagation Round algorithm are tested:

The *Normal Propagation Round* algorithm takes all permutations of possible correspondence assignments between the neighbors $n(v_x)$ and $n(v'_y)$, even if the number of neighbors differs. The combined similarity is then the mean of the similarities of the matched neighbors plus the similarity of the original vertex pair $n(v_x)$ and $n(v'_y)$ divided by two.

Suppose $s(v_x, v'_y)$ is the similarity between two vertices. M is the mapping for vertices $M(n)$ and $M'(n)$ from G to G' where $M(n) \in n(v_x)$ and $M'(n) \in n(v'_y)$. For one specific mapping M the combined similarity is

$$s_{combined}(v_x, v'_y, M) = \frac{s(v_x, v'_y) + \sum_{n=0}^{|M|} \left(s\left(M\left(n\right), M'\left(n\right)\right)\right)}{2}.$$

The *Strict Propagation Round* algorithm takes the topological constraints of the graph into account. If the number of neighbors differs, the similarity value for two vertices v_x and v'_y is heavily penalized without looking at the neighbors. In case of an equal number of neighbors ($|n(v_x)| = |n(v'_y)|$), only those mappings are checked that are in the correct geometrical order. As explained before in Section 3.7, the edges leading to neighbors are maintained in a ring buffer that is sorted by the relative angle of each edge. The set of neighbors $n(v_x)$ is sorted in the same way, i.e., using the order of the edges that lead to them, hence providing a very efficient access to the neighbors left or right of another neighbor.

A correct geometrical order implies that a mapping from $n(v_x)_i$ to $n(v'_y)_j$ is only valid if also $n(v_x)_{i+1}$ is mapped to $n(v'_y)_{j+1}$. The total number of permutations that comply to this restriction is simply the number of neighbors. This is because there are $|n(v'_y)|$ partners to choose for the first vertex in $n(v_x)$, but afterward all further mappings are determined by their topological order determined by their angles.

The complexity of the propagation rounds algorithm depends on the number of vertices in the graph $|V|$ since we are calculating the new similarity for every vertex. It also depends on the number of neighbors of the vertices, which is factorial in the number of neighbors in the case of a *Normal Propagation Round* and linear for

the *Strict Propagation Round*. But since there are typically very few neighbors for every vertex (typically three or four) and never really many, one can actually regard this as a bounded constant, such that in the end the overall propagation round algorithm has a complexity of $O(|V|)$.

4.2 Local Vertex Similarity

The Local Vertex Similarity s_l is a similarity approach purely based on information from the Topology Graph. It averages over a number of dissimilarity checks to determine the dissimilarity between two vertices:

- First, it is checked if the two vertices differ in their type, which can be either *Spurious* for spurious dead-ends, *DeadEnd* for regular non-spurious dead-ends, *RayEnd* for the end vertices of rays, or *Junction* for vertices with a degree larger than two.
  If both vertices have the same type, the value of Δ_{Type} is 0. If they differ, it is a 1.
- The second dissimilarity check Δ_{Exits} takes the difference of the number of neighbors into account, with a score of 0 if the number of neighbors is equal and a score of 1 if they differ.
- The third value is obtained from the average of four other sub-scores describing the vertex:
  - The first and second sub-scores use the values of the biggest ($\Delta_{anglesMax}$) and smallest ($\Delta_{anglesMin}$) angle between neighboring edges of a vertex. The difference of those angle-differences for the two to be compared vertices is used together with the parameters $\sigma_{angleOff}$ and $\sigma_{angleMax}$ to calculate the scores.
  - The third sub-score ($\Delta_{obstacle}$) compares the distance from the vertex to the closest obstacle for both vertices. In case this vertex is a joined vertex, those distances are averaged for all merged vertices that form this vertex. The related parameters are $\sigma_{obstacleOff}$ and $\sigma_{obstacleMax}$.
  - The fourth sub-score is only applied to joined vertices (Δ_{joined}). It uses the summed up distances from the original vertices to the joined vertex. Those according values of the two joined vertices are compared using $\sigma_{joinedOff}$ and $\sigma_{joinedMax}$.

The first, second and third - which is itself an average of four sub-scores - score values are averaged, i.e., the Local Vertex Similarity $s_l()$, respectively the corresponding dissimilarity $\bar{s}_l()$ is:

$$\bar{s}_l(v,v') = \frac{1}{3} \cdot (\Delta_{type} + \Delta_{Exits} + (\Delta_{anglesMax} + \Delta_{anglesMin} + \Delta_{obstacle} + \Delta_{joined})/4)$$

4.3 Enhanced Vertex Similarity

The simple Local Vertex Similarity presented in the previous section uses only information from each vertex in the Topology Graph. The Enhanced Vertex Similarity takes in addition information from the neighbors of each vertex into account, i.e., it follows the idea of the Propagation Round. In addition, the Enhanced Vertex Similarity takes further graph properties into account.

The Enhanced Vertex Similarity is defined as a combination of the best exit assignment $EA()$, which is introduced in the following subsection, and the Local Vertex Similarity. We found the $EA()$ in practice to be slightly more performing than the Local Vertex Similarity $s()$; it is hence slightly higher weighted when both are combined:

$$\bar{s}_e(v,v') = \frac{3 \cdot max(EA(v,v')) + 2 \cdot \bar{s}_l(v,v')}{5}$$

4.3.1 Exit Assignment Similarity

Given a correspondence assumption between two exits of two supposedly corresponding vertices - or short an assignment between exits - we are interested in computing a (dis)similarity of the two exits. As mentioned before, one problem is that there are - as long as there is more than one exit - different ways to assign the exits of the two vertices to each other. As a solution, all permutations of exit assignments between two vertices are generated:

The mapping em of one exit ex_v from vertex v from graph G to one exit $ex'_{v'}$ from graph G' is a 2-tuple: $em = (ex_v, ex'_{v'})$.

An exit assignment $ea(v,v')$ is a set of exit mappings em. It has $min(deg(v), deg(v'))$ entries. No exit appears twice in this set:

$$\forall em \in ea, \nexists em^* \in ea \mid (em_1 = em^*{}_1 \wedge em_2 \neq em^*{}_2) \vee (em_2 = em^*{}_2 \wedge em_1 \neq em^*{}_1)$$

$EA(v,v')$ is the set of all possible permutations of exit assignments $ea(v,v')$.

The exit assignment dissimilarity is calculated on an exit assignment $simi_{ea}$. First, the angle dissimilarity $as(ea)$ is calculated. It looks, for every exit, at the

angle to the next exit (clock wise). The difference of this angle for the paired exits is calculated into a score using $\sigma_{obstacleOff}$ and $\sigma_{obstacleMax}$ for all the assigned exits. Then the scores for those angle-differences are averaged.

The exit assignment dissimilarity uses a combination of the pair-wise Exit Similarity $pe(ex, ex')$ presented next and the angle dissimilarity $as(ea)$:

$$simi_{ea}(ea) = \frac{2 \cdot \sum_{n=0}^{|ea|} \left(pe(ea_{n_1}, ea_{n_2})\right) / |ea| + as(ea)}{3}$$

The pair-wise Exit Similarity $pe(ex, ex')$ calculates the dissimilarity between the exits ex and ex' and consists of two equally weighted parts. First, there is a comparison of the properties of the edges and then of the dissimilarity of the target vertices. Note that a dead-end vertex is marked as spurious if the related edge is short and it hence may be caused by sensing artifacts - regular non-spurios dead-ends as well as all other vertices like junctions are denoted as major vertices.

For the edge comparison the scores of the following attributes are averaged:

- $\Delta_{nextVertex}$: Difference in the distances to the next vertex.
- $\Delta_{nextMajorV}$: Difference in the distances to the next major vertex.
- $\Delta_{obstacleMin}$: Difference in the distances to the closest obstacle along the edge.
- $\Delta_{obstacleAvg}$: Difference in the average distances to obstacles along the edge.
- $\Delta_{existsSpurious}$: Difference in the presence of at least one Spurious vertex along the exit.

The dissimilarity for the target vertices ts is set to be 1 (worst) if exactly one of the edges is a ray and 0 if both are rays. Otherwise the local vertex dissimilarity (see 4.2) of the target vertex and the next major vertex are used. $target(ex)$ returns the target vertex given an exit. If this edge is a loop, the target vertex will be the same vertex as the start vertex.

$$ts(ex, ex') = \begin{cases} 0. & \text{if } ray(ex) \wedge ray(ex'); \\ 1. & \text{if } ray(ex) \text{ xor } ray(ex'); \\ s_l(target(ex), target(ex')) \end{cases}$$

$$pe(ex, ex') = \left(\frac{1}{8} \cdot (\Delta_{nextVertex} + \Delta_{nextMajorV} \cdot 4 + \Delta_{obstacleMin} + \Delta_{obstacleAvg} + \Delta_{existsSpurious}) + ts(ex, ex')\right)/2$$

4.4 Sensor-based Vertex Similarities

The idea behind sensor-based methods for vertex similarity is to compare the local appearance of the grid maps around the two vertices to generate a similarity value, i.e. to use a more classical place recognition to estimate vertex correspondences. This involves two aspects. First, the according parts of the map have to be extracted. Second, those parts have to be compared. For the extraction, two strategies are investigated, namely Disk-Extraction and Raytrace-Extraction, which extract a local circular template, or simulate a local omni-directional range sensor view, respectively. Different registration algorithms are tested for the comparison of the extracted map patches, namely two variants of the Iterative Closest Point (ICP) algorithm [5]) and the improved Fourier Mellin Invarinat (iFMI) algorithm [24, 7].

Note that just the local part of the map around the vertex location is used. Therefore, errors in other parts of the map do not effect the similarity of a vertex with its counterpart in the ground truth map. Both extraction approaches have a radius as parameter, specifying the disk from which obstacle points from the map are extracted into a 2D point cloud. The Disk-Extraction takes all points within this radius and adds it to the point cloud. The Raytrace-Extraction only takes the obstacles that are visible from the location of the vertex, thus omitting obstacle points that are within the radius but that are obstructed by closer obstacles. See Figure 8 for examples of both. The idea behind the latter approach is to only use obstacles from a room and not from the area behind a wall. This makes the point cloud even more local and thus localization errors effect vertex similarities based on this extraction methods measure even less. Also, the number of points generated is usually much less compared to the Disk-Extraction, which can lead to faster registrations. On the other hand, since there are less points, it might also be less descriptive and it can turn out to be less robust in the place recognition.

The similarity of the two vertices has to be independent of the rotation of the point clouds, because errors in the initial orientation of the maps or localization errors might lead to (significantly) different orientations. Therefore, registration algorithms should provide an inherent rotational invariance in the comparison step.

4.4.1 ICP-based Vertex Similarity

The Iterative Closest Point (ICP) is a classical algorithm to calculate the transformation between two point clouds [5]. Starting with an initial assumption for the

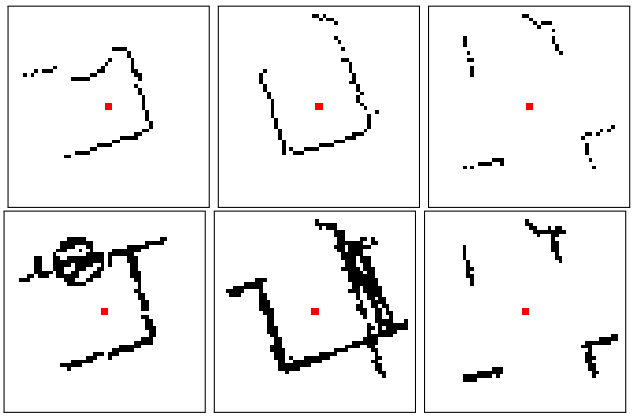


Fig. 8 Example extractions for the vertices 2, 12 and 22 (from left to right) from Vertex Similarity Experiment Map H (see Figure 9). On the top the Raytrace-Extraction and on the bottom the Disk-Extraction is shown. The location of the related vertex is marked with a red dot.

transformation, it calculates the transformation with the lowest mean squared error between nearest neighbors from the two point clouds. It then iterates the process with the found transformation as new initial transformation until either a maximum of iterations or a threshold for the mean squared error or its change is reached.

The first ICP Similarity approach (called *Simple ICP Similarity*) assumes the identity for the initial transformation and uses the square root of the resulting mean squared error as similarity value. The idea is, that this value will be lowest for the vertices from the two maps that are on the same location and higher for all (or at least most) of the other candidates.

In general, ICP is quite sensitive with respect to the initial guess of the transformation between the two point clouds. If this assumption is off the actual transformation by a larger amount, ICP often is caught in a local minimum. The Simple ICP Similarity thus depends on the orientation of the maps and it is thus not perfectly rotation invariant, as required. As can be seen in the experiments (see Section 4.5) this leads to unsatisfactory results.

The *Rotated ICP Similarity* mitigates this problem by running the algorithm a couple of times with initial guesses for the rotation that are evenly distributed over the 360 degrees of a circle. For example, with a parameter n of 12 for the number of differently seeded ICP runs, it increments the initial start angle by 30 degrees for each trial. Thus the correct rotation between the two point clouds will not be off more than 15 degrees from the initial guess of the best fitting trial. This gives a very good chance that the ICP run with a starting rotation close to the correct value will not be caught in a local minimum but advance to the global minimum, thus giving the correct and lowest mean squared error. The square root of the lowest of the mean squared errors for all runs is then selected as the similarity value. Obviously the main disadvantage of this approach is the increase in computation time (n-times).

4.4.2 iFMI Similarity

The improved Fourier Melling Invariant (iFMI) algorithm [8,7] is a further option to calculate the similarity between the two point clouds of obstacles from the area around the respective vertices from the two maps. First, the point clouds have to be put into a square 2D grid as iFMI works on square matrices. The iFMI is an improved version of the well-known Fourier-Mellin image registration technique, which is inherently rotation invariant. It is hence well suited for our purposes here. The signal to noise ratio around the Dirac pulse of the translation step indicates the quality of a match and it is hence used here as the similarity value.

4.5 Vertex Similarity Experiments

The performance of the different vertex similarity algorithms is evaluated now in experiments. The different approaches are applied to a number of maps that show the same environment - the random maze from the Response Robot Evaluation Exercise at Disaster City 2010. For the experiments nine different maps are used. One example map (Map H) is shown in Figure 9 - all other maps are shown in Figure 10. Map A is a ground truth map, while maps B, C, D and E where generated by different mapping algorithms. All those maps are roughly in the same orientation. Map F is the Map B rotated by exactly 90 degrees, while Map G is Map B rotated by 180 degrees. Map H is Map B rotated by approximately 140 degree and Map I is Map C rotated by about 37 degrees.

The maps show the Topology Graphs and numbers for the vertices. The numbering simply starts at the top left and goes down to the bottom right.

Since the maps all show the same environment, it is expected that the Topology Graphs are very similar and this is indeed the case. This is used to assess the performance of the vertex similarity experiments. The vertices from two maps at corresponding locations should have very good similarity values, i.e., high scores, while non-corresponding vertex pairs should have lower values. The evaluation criterion in these experiments is hence simple, namely whether for any given vertex from one map, the best (highest vertex similarity value) of all vertices from another map is the one corresponding

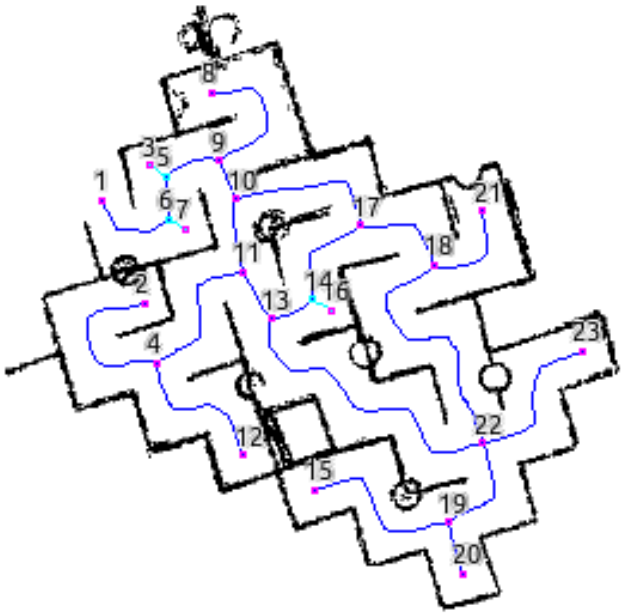


Fig. 9 Vertex Similarity Experiment Map H

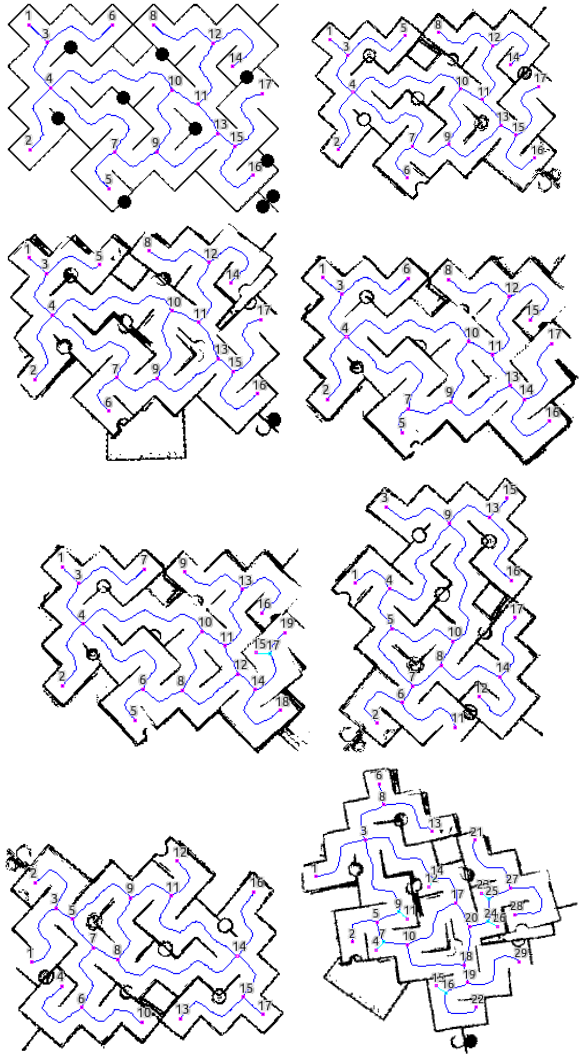


Fig. 10 Vertex Similarity Experiment Maps A (top left), B (top right), C, D, E, F, G, and I

to that location. For this purpose the correctly corresponding vertices between the maps were determined by hand for comparison.

For example, the mapping from Map A to Map H is: (1,20) (2,23) (3,19) (4,22) (5,21) (6,15) (7,18) (8,12) (9,18) (10,13) (11,11) (12,4) (13,10) (14,2) (15,9) (16,8) (17,1).

One can see that, for example, the vertex number 1 in Map A corresponds to the vertex number 20 in Map H. So in the experiments, vertex number 1 in Map A will be compared with all the vertices from Map H, using one of the vertex similarity methods. If the vertex from Map H with the lowest (best) value is vertex number 20, then this was a success, otherwise a failure.

All maps feature 17 vertices that always appear at the same location, so those are the ones used for the success calculation. Some maps have some additional Spurious vertices (e.g. for Map H vertices 3, 5, 6, 7, 14 and 16) that are included in the similarity calculations (and could thus have the lowest similarity value for another vertex, making this comparison a failure), but that do not otherwise effect the success calculation.

Not all combinations of maps were used. This is because the Topology Graphs for maps that are just rotated by 90 or 180 degrees are exactly the same. Thus the Simple and Enhanced Vertex Similarity have a 100% success rate. This seems like an unfair advantage to those algorithms, since in reality the exact same maps will not be used. Note that maps rotated at an angle other than 90, 180 or 270 degree always have differences due to the interpolation and thus the Topology Graph differs to some extend.

Furthermore, a distinction for map pairs with a similar orientation and rotated maps has been made to show the effects of the rotation for the different algorithms. The success rate is averaged over all vertices and map pairs for the “No Rotation” case, for the “Just Rotation” case and for the combined results called “Both”.

The 10 combinations used for “No Rotation” and the 23 map pairs for ”Just Rotation“ are:

- No rotation (10): AB, AC, AD, AE, BC, BD, BE, CD, CE, DE
- Just Rotation (23): AF, AG, AH, AI, BH, BI, CF, CG, CH, CI, DF, DG, DH, DI, EF, EG, EH, EI, FH, FI, GH, GI, HI

4.5.1 Experimental Evaluation of the Vertex Similarity Algorithms

In total 38 methods to compute a vertex similarity are tested during the experiments. The *Local Vertex Similarity*, the *Enhanced Vertex Similarity*, the *Simple ICP Similarity*, the *Rotated ICP Similarity* (abbreviated as ICP-12), and the *iFMI Similarity* are used. To all of them two *Propagation Rounds* are applied - both strict (abbreviated as _1_s and _2_s) as well as normal ones (abbreviated as _1 and _2) are used. For the Enhanced Vertex Similarity only one Round is applied, since it already contains a round in its default version. The appearance-based algorithms (ICP and iFMI) were tested with both *Raytrace-Extraction* and *Disk-Extraction* (those results are shown in different diagrams).

All 38 algorithms have been applied to all 33 map pairs. The ICP algorithms use a maximum number of 100 iterations and an error reduction factor tolerance of 0.01. The radius for the Raytrace- and Disk-Extraction is 50 pixel or 2.5 meter. The resolution for the iFMI algorithms is accordingly 100x100 pixel. The average point cloud size for the Raytrace-Extraction for both maps combined is 76 points, while there are 619 points on average in the Disk-Extraction.

4.5.2 Configuration Parameters for Vertex Similarity

The following parameters are used in the calculation of the Local Vertex Similarity:

- $\sigma_{angleOff} = 1$ degree. The biggest and the smallest angle between edges of a vertex are used as descriptors. When comparing these descriptors between two vertices, this parameter is used to compute the similarity value, i.e., we assume angles within this tolerance to be the same.
- $\sigma_{angleMax} = 12$ degree. This parameter limits the similarity in the angle descriptors, i.e., from that threshold on, angular descriptors are considered to be different to each other.
- $\sigma_{obstacleOff} = 0.5$. The average distance to the nearest obstacle is a descriptor for edges and vertices. When comparing these descriptors, this parameter describes the tolerance when the distances are still considered to be the same.
- $\sigma_{obstacleMax} = 2$. When the difference in the average distances to the nearest obstacle exceeds this value, the related descriptors are considered to not be similar anymore.
- $\sigma_{joinedOff} = 0.5$. For vertices that were created by joining several vertices in the merging step, the average distance to the joined vertices is used as a descriptor. This parameter corresponds to the tolerance when these descriptors are still considered to be identical.
- $\sigma_{joinedMax} = 2$. This parameter accordingly sets the threshold to consider descriptors from joined vertices as different.

As mentioned before, the offset values are tolerances to small differences in the descriptor values, i.e., up until the offset, the descriptors are considered to be exactly the same with a similarity score of 1. The maximum values are thresholds to determine when descriptors differ, i.e., from that value on, the descriptors are considered to be completely different with a similarity score of 0. In between these parameters, the similarity is normalized to $]0, 1[$.

As mentioned before, note that all spatial parameters are unit-less, i.e., they are in the scale of the grid map. Given the "geo-refencing" information, it can make sense to relate them to metric values. For example, if $\sigma_{obstacleMax} = 2$ and the resolution of the maps is 5cm, then this value corresponds to 10cm.

	time (msec) 17x17 similarities
Local Vertex Similarity	0.6410
Enhanced Vertex Similarity	11.2360
Raytrace-Extraction (R.E.)	131.5789
Disk-Extraction (D.E.)	128.2051
Simple ICP (R.E.)	232.5581
Rotated ICP Similarity (R.E.)	2,439.0244
Simple ICP (D.E.)	2,702.7027
Rotated ICP Similarity (D.E.)	26,315.7895
iFMI	3,225.8065
One Propagation Round	0.0053
One Strict Propagation Round	0.0051

Table 1 Runtimes of the different algorithms (R/D.E. = Ray-/Disk-Extraction).

4.5.3 Experimental Results

The 17 x 17 vertices from any of the two map combinations have 289 possible vertex correspondence pairs, so there are 578 extractions needed (2 per pair). Table 1 shows the runtimes for the different algorithms for the complete computation of the 17 x 17 similarity values for each. The computation was done in a single thread on a 2.8 GHz Intel Core 2 processor.

Note that any of the vertex similarity methods can be followed by one or more (Strict) Propagation Rounds. As every (Strict) Propagation Round uses the precomputed similarities for consistency checking, its runtimes are fixed and independent of the vertex similarity method with which it is combined. Also note its very high speed.

The extraction times are included in the ICP and iFMI runtimes. The main result of the runtime comparison is, that the calculation of a propagation round is about 2 orders of magnitude faster than the Local Vertex Similarity which in turn is one order of magnitude faster than the Enhanced Vertex Similarity. Nevertheless, all this is still two orders of magnitude faster than the sensor-based approaches.

Figure 11 shows the results of the similarity experiments. The higher the bars the better the similarity. Both graphs show the results from the Local and the Enhanced Vertex Similarities, including the versions with Propagation Rounds. These are twice the same results - they are included in both graphs to ease the comparison with the different sensor-based methods. The graphs show the different results for the sensor-based approaches (ICP and iFMI) with the upper graph (11(a)) showing the results while using the Raytrace-Extraction and the lower graph (11(b)) showing the results based on the Disk-Extraction.

Note that the accuracy percentages shown here are achieved with the simplest matching algorithm possible, namely just matching a vertex against the one vertex from the other map with the best similarity score. Later on, better methods of matching will be introduced. The values computed using the similarity algorithms presented in this section are then used to guide this search and thus do not need to be perfect.

4.5.4 Discussion of the Results of the Vertex Similarity Experiments

First the effects of the Propagation Rounds are evaluated. It can be seen that for most approaches adding Propagation Rounds improves the result. Often the Strict Rounds approach is better than using Normal Rounds. The Enhanced Vertex Similarity, which basically is a Local Vertex Similarity with one Propagation Round and additional constraints such as the edge lengths, is superior to the Local Vertex Similarity with one Propagation Round, proving that the additional information is quite helpful. The Enhanced Vertex Similarity is already as strict as a Strict Propagation Round and therefore there is very little difference between the additional Normal and the additional Strict Propagation Round.

A very clear result is that the range-sensor-based place recognition (ICP and iFMI) perform much better using the Disk-Extraction than with the Ray-Extraction. It seems that raytracing omits too much information and is just too sparse.

The effects on the similarity algorithms of a rotation of or in the map can be seen when comparing the "No Rotation" bars with the "Just Rotation" ones. One can see that the graph based approaches (Local and Enhanced Vertex Similarity) have very similar values for both cases - this is especially true for the Enhanced Vertex Similarity. The place-based solutions have more problems. The Simple ICP Similarity is for the "No Rotation" cases relatively decent and for the Disk-Extraction even excellent, but for the "Just Rotation" combinations it performs very badly. Only for the information-rich Disk-Extraction the Rotated ICP Similarity is not significantly affected by the rotations. The same is true for the iFMI approach.

The best results are clearly achieved using the Rotated ICP Approach with Disk-Extraction. Adding one strict Propagation Round seems to improve the result - but just by a little bit. With or without any kind of Propagation Rounds - the values are close to or above 90 Percent, regardless of the rotation. If for any application the runtime is also important, the Enhanced Vertex Similarity with one additional Propagation Round is also very interesting. The values are in the high eighties while this algorithm is about 2000 times faster (three orders of magnitude) then the Disk-Extracted Rotated ICP method.

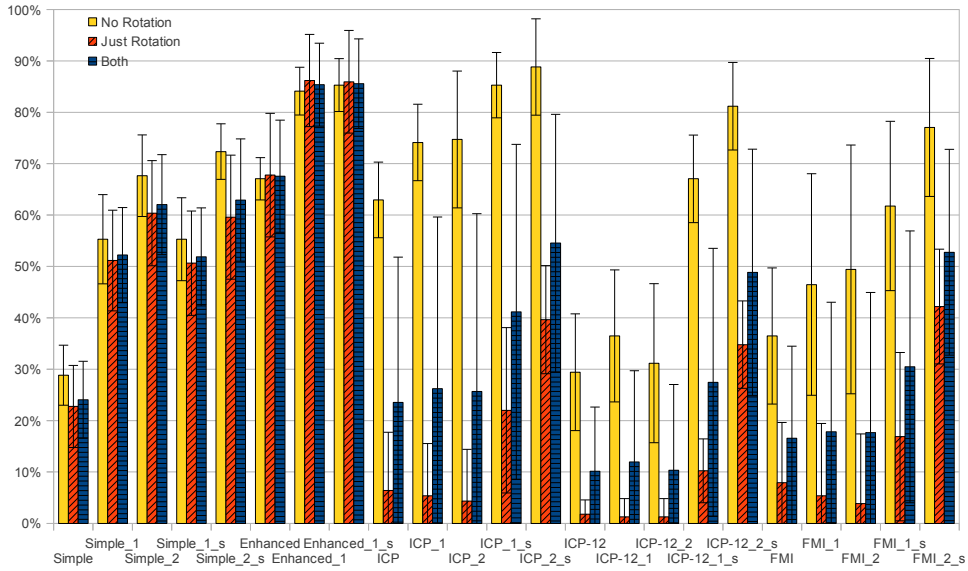
5 The Matching of two Topology Graphs

Now the matching between two graphs can be determined. In the following, the smaller graph, i.e., the one with less vertices, of two graphs is denoted as first graph G, and the larger one is denoted as second graph G'. If not mentioned otherwise, G is matched against G', which is typically a robot-generated (partial) map of the environment that is matched against an exhaustive ground truth representation.

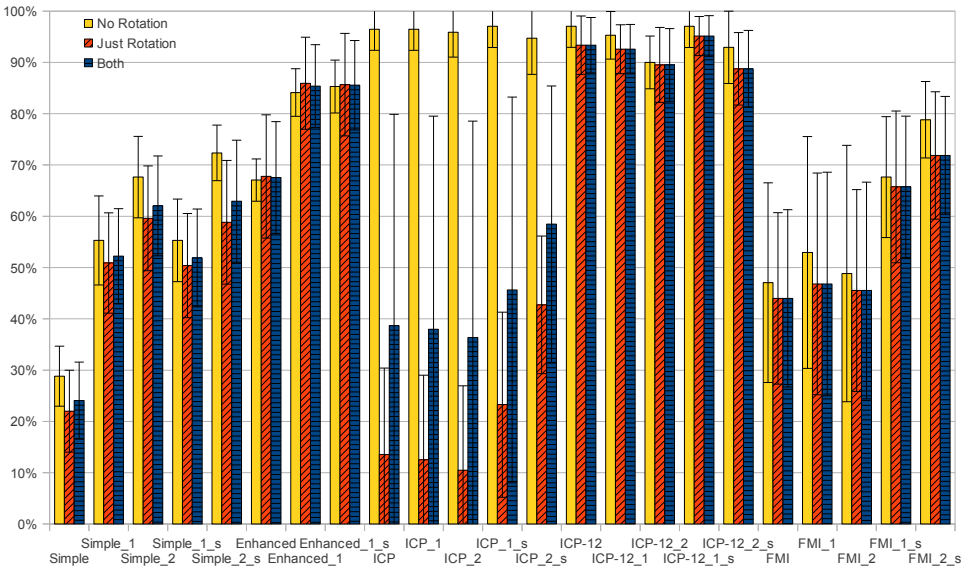
Two Topology Graphs are matched by first finding an isomorphism and then applying a recursive neighbor growing approach. Finding an isomorphism is a quite fast and strict way to find parts of the maps that do not diverge in the connections of the graph. The second, more heuristic approach is the recursive neighbor growing algorithm which makes heavy use of the vertex similarity. It mostly ignores the constraints that guide the isomorphism algorithm and uses a similarity value for the whole match to guide the search.

5.1 Finding Isomorphisms

A graph isomorphism is a basic concept of graph theory [37,10]. For map matching we are looking for subgraphs from both graphs that form an isomorphism, which



(a) Using the Raytrace-Extraction for the Sensor-based Approaches.



(b) Using the Disk-Extraction for the Sensor-based approaches - the values for the topological methods are the same as the above values and they are just included for better comparison.

Fig. 11 Combined average success rates and standard deviations of the Vertex Similarity experiments

is known as the maximum common subgraph isomorphism problem. In our application, we deal with planar graphs (no edges intersect on a 2D plane), we have a very good heuristic for possible vertex correspondences (the Vertex Similarity) and we have local metric information like edge orientations that can be exploited. Due to these special conditions, we can use a very efficient custom approach to solve the graph matching problem at hand.

Concretely, just two simple rules are used for finding the isomorphism:

- the number and the geometric order of the edges of the matching vertices should match
- and the length of the corresponding edges should coincide to at least a certain degree.

5.1.1 The Vertex Assignment Structure

The vertex assignment structure maintains the information about the assignment of possible correspondences of vertices and exits of the two graphs that are compared. The mapping between vertices is stored as two hash maps for both directions for quick access. An other mapping is used for the Exits that are already assigned.

The similarity value for the whole assignment is the sum of the similarities of the already assigned vertex pairs using one of the vertex similarity algorithms presented above.

Using this structure it is possible to efficiently check if a vertex has already been assigned to a vertex from the other graph - for both graphs respectively. One can also check if assigning a vertex $v \in G$ to a vertex $v' \in G'$ would be consistent: if both vertices have not yet been assigned then the result is true by default. If one of them has been assigned and the other was not, then the result if false by default. If both have been assigned, then the result is true if they have been assigned to each other and false if they have been assigned to different vertices.

5.1.2 Finding Candidate Isomorphisms

The pairwise vertex similarities allow to determine a straightforward candidate subgraph isomorphism by assuming correspondences between the most similar vertices. The goal is now to find possibly larger subgraphs between which a proper isomorphism can be established.

For every vertex v_x for which no correspondence has not yet been assigned, it is attempted to find a new isomorphism starting at said vertex v_x. This is done by matching this vertex with every vertex from the other graph that has the same number of connected edges - starting with the ones with the highest similarities. The recursive Wavefront Propagation algorithm presented in the following subsection is then applied to the pair of best corresponding vertices, i.e., the two most similar ones. When the Wavefront Propagation ends with a complete subgraph isomorphism as result, it is checked if the number of matched vertices in this assignment is at least as big as the configuration value $\sigma_{isoMinVert}$. If this is the case the result is accepted. The biggest accepted assignment for v_x is then added to the final assignment.

5.1.3 Wavefront Propagation

In the Wavefront Propagation algorithm, vertices that have edges to other, not yet matched vertices form the wavefront. In the beginning, the best correspondence pair is in this wavefront, i.e., the best pair from the candidate vertex list (Section 5.1.2). Although the Wavefront Propagation is a recursive algorithm by design, most matches of vertices will not meet the strict parameters for a Topology Graph isomorphism, such that mainly a computationaly very efficient iterative search is performed.

Algorithm 5 The Recursive Wavefront Propagation

```
function recursiveWavefrontPropagation( frontier, currrentVertexAssignment )
while not frontier.empty() do
  currentVertexPair = frontier.pop()
  goodAssignments = getGoodAssignments( currentVertexPair, currrentVertexAssignment )
  if goodAssignments.size() == 1 then
    addToVertexAssignmentAndFrontier( goodAssignments.top(), frontier, currrentVertexAssignment )
    continue
  else if goodAssignments.empty() then
    continue
  end if
  // goodAssignments has more than one entry - recursion!
  for all assignment in goodAssignments do
    frontierCopy = frontier
    assignmentCopy = currrentVertexAssignment
    addToVertexAssignmentAndFrontier( assignment, frontierCopy, assignmentCopy )
    recursiveWavefrontPropagation( frontierCopy, assignmentCopy )
  end for
end while

function getGoodAssignments( currentVertexPair, currrentVertexAssignment )
rtnVarGoodAssignments.empty()
exitAssignmentTable = vertexTable [currentVertexPair.first] [currentVertexPair.second].exitTable
for all exitAssignment in exitAssignmentTable do
  if checkExitAssignmentForConsistency( exitAssignment, currrentVertexAssignment ) then
    rtnVarGoodAssignments.insert( exitAssignment )
  end if
end for
return rtnVarGoodAssignments
```

In Algorithm 5 the exit assignment table, which is generated during the calculation of the Enhanced Vertex Similarities (see Section 4.3), is used in the function `getGoodAssignments`. Each exit assignment is the pairwise matching of exits of one vertex from the first graph with the exits of the matched vertex from the second graph. The exit assignment table consists of all permutations of matches between the exits of the vertices.

The function "checkExitAssignmentForConsistency" checks if an exit assignment is consistent with the current vertex assignment. This is only true if the exits are equal in number and are matched in the correct

(local geometric) order. Otherwise the function immediately returns false. Then it checks if the vertices that the edges are leading to are already matched and if so, if they are matched with the correct vertices, using the Vertex Assignment Structure from above. In this step the possibility that an edge might lead to a Spurious vertex is taken into account. The function thus also returns consistency if a Spurious vertex matches a major vertex - as well if major vertices match. Additionally, the length of the edges are compared. The following formula to compute the similarity value *ratio* is used to be more lenient on shorter edges:

$$ratio(\text{firstDistance, secondDistance}) = \frac{\text{firstDistance} + \sigma_{isoEdgeDistMinAllow}}{\text{secondDistance} + \sigma_{isoEdgeDistMinAllow}}$$

$$if(ratio(...) > 1) \quad ratio(...) = 1/ratio(...)$$

If *ratio* is then smaller than $\sigma_{isoEdgeDistFactor}$ this exit assignment is rejected.

5.2 Grow Neighbors

The grow neighbors algorithm is quite similar to the isomorphism algorithm from above, but it is a more heuristic version with more relaxed constraints that can be used in addition to find a few more correspondences. Though there is the risk of false positives due to its more heuristic nature, we found it to be very robust in practice.

It uses a recursive wavefront approach to extent the match according to the edge assignments that are possible for a match in the wavefront. Unlike the proper isomorphism method, the edge assignment does not have to be in order and also vertices that differ in their numbers of edges are expanded. The consistency check only returns false if an inconsistency in the vertex assignments is detected.

The vertex similarity is used extensively in this method. If a new match is found it is checked if the similarity of the whole match falls below a certain threshold. If that is the case the match is not added. Figure 12 shows an example match between the ground truth map from RoboCup 2010 in Singapore and a robot generated map using the isomorphism and neighbor growing approaches.

5.3 Configuration Parameters for Graph Matching

The following configuration values are used as parameter in the graph matching in the experiments described here and in the next section:

- $\sigma_{isoMinVert} = 6$. During the isomorphism algorithm, this value is the minimum number of vertices that have to be matched so that a matched subgraph is accepted and kept as a result. We consider values between five and eight to be useful for this parameter; higher values can be beneficial if there are repetitive topologies in the environment.
- $\sigma_{isoEdgeDistMinAllow} = 20$. During the isomorphism algorithm edges are only matched if their distance is compatible. If the length differences are smaller this threshold-parameter, they are considered to be similar.
- $\sigma_{isoEdgeDistFactor} = 0.6$. If the length difference of two edges relative to the length of the shorter one exceeds this threshold-parameter, they are considered to be not compatible.

As mentioned before, note that all spatial parameters are unit-less, i.e., they are in the scale of the grid map. Given the "geo-refencing" information of the map, it can make sense to relate them to metric values. For example, if $\sigma_{isoEdgeDistMinAllow} = 20$ and the resolution of the maps is 5cm, this value corresponds to 100cm.

6 Map Evaluation using Matched Topology Graphs

Once the matching of the Topology Graph G of a map with the Topology Graph G' of a reference map is done, the actual evaluation of the map quality is quite easy. It can for example be done similar to the approach of the Fiducial Map Metric [25] - but with vertex-correspondences from the Topology Graphs instead of requiring fiducials as artificial landmarks. For a map quality assessment, the map attributes can hence be calculated as follows:

- Coverage: Calculate the percentage of vertices from the ground truth Topology Graph matched to vertices from the graph of the robot map.
- Global Accuracy: Correctness of the locations of the matched vertices in the global reference frame.
- Relative Accuracy: Correctness of locations of matched vertices after correcting (the initial error of) the map reference frame using Horn's algorithm [12].
- Local Consistencies: The shortest distances between any two vertices in the ground truth graph are calculated. All vertex pairs are permutated. The pairs

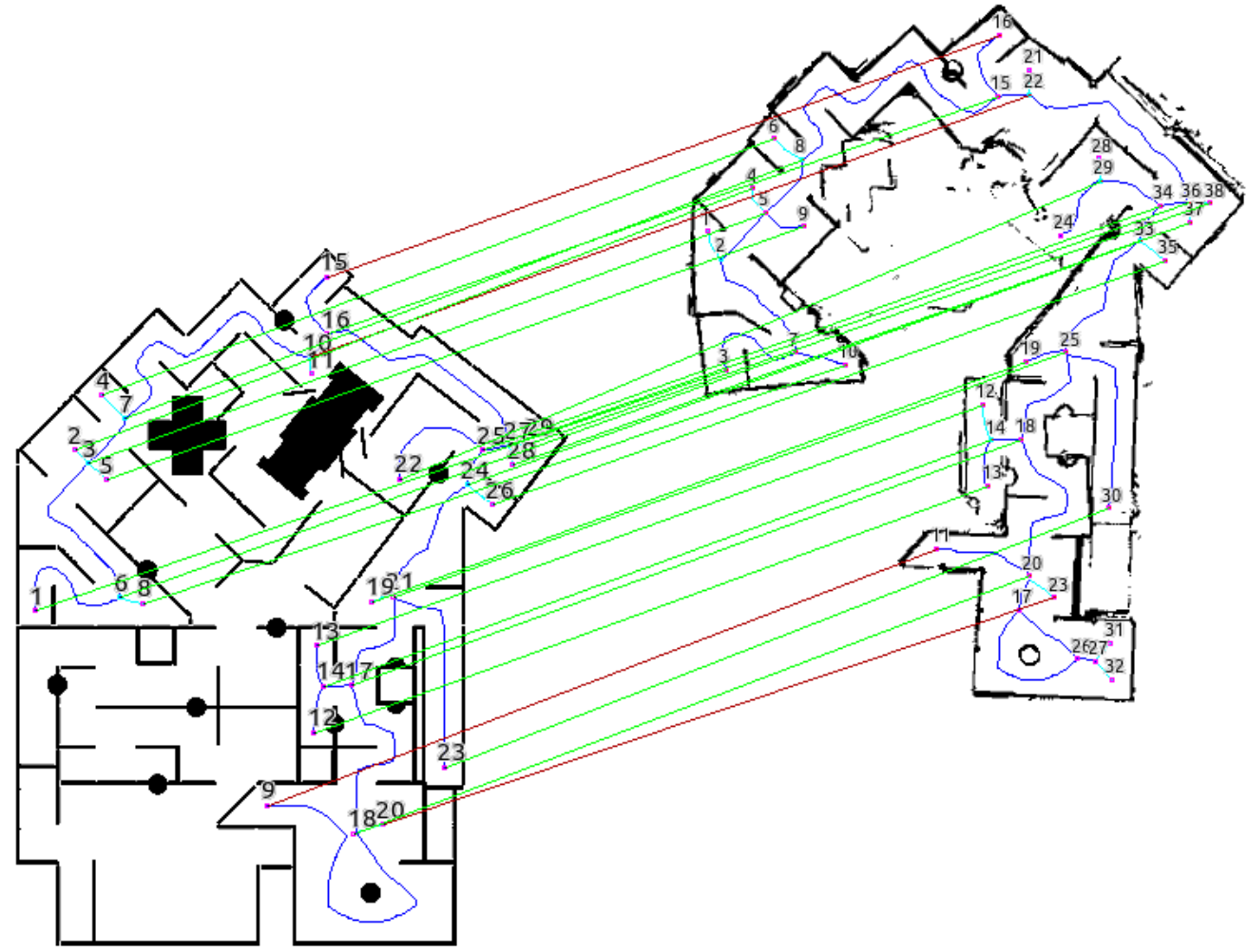


Fig. 12 Example match between a ground truth map (left) and a robot generated map (right) using the isomorphism algorithm (green lines indicate the correspondences found), followed by the grow neighbors approach that finds a few additional correspondences (red lines). There are 28 assignments made and one assignment is missing from the left graph (vertex 11). This is the RoboCup environment from Figure 1.

are put into different classes of difficulty according to their distance over the graph. The geometric distance between the two vertex positions is calculated and compared to the geometric distance between the two matched vertices from the robot generated map. The errors of those comparisons are averaged for each of the classes of difficulty.

6.1 Calculating the Brokenness

In [6] another map attribute, the brokenness, is proposed. It is defined as "the degree with which a map can be partitioned into regions that are locally consistent with ground truth but off relative to each other." Calculating the brokenness using a Topology Graph that is matched against a ground truth Topology Graph is relatively simple. A subset of connected vertices, a *Brokenness Group*, has to be found whose minimal mean square error regarding the location of the vertices compared to the matched ground truth vertex locations does not exceed a certain threshold. This minimal mean square error is easily computed using Horn's algorithm. This subset is saved. The process is iterated over the remaining vertices that are not yet part of a subset. A subset is only accepted if it contains at least a certain number of vertices. The number of subsets found minus one is then the brokenness degree as defined by Birk.

6.1.1 Finding Brokenness Groups

The goal is to find a set of vertices that geometrically fit to each other. For this, two thresholds are defined. The $\epsilon_{brokenThres}$ describes the upper bound on how big the

Mean Squared Error of the vertex locations with regard to the matched ground truth vertices can be. And the $\epsilon_{brokenMinV}$ is the minimum number of vertices needed in this set to be considered a Brokenness Group.

The idea behind the brokenness concept is, that the part of a map that is broken has in itself no substantial local errors. Maps are typically broken because of either bump noise (the wheels/ tracks slip on the ground, maybe because the robot hit an obstacle, thus the odometry is broken) or because of a failed registration of the sensor data. Even with the use of SLAM algorithms according large local residual errors can occur. Areas of maps where the mapping consistently failed and which thus show no large-scale identifiable structures are not considered broken areas. Since broken parts are internally relatively intact, the Topology Graph should match nicely, too. This idea guides the implementation for the brokenness search, for which a Wavefront Propagation is used here.

We start with one matched vertex pair (v_x, v'_y) from G and G'. The neighbors of this match have to be added to the frontier of to be tried matches.

6.1.2 Frontier Adding

The incident vertices (neighbors) of v_x from graph G are iterated. If those were matched to a vertex from G' during the Topology Graph Matching, they are about to be added to the frontier. But before that, it is checked that this vertex match is not already in another Brokenness Group, and not in the current Brokenness Group, nor already in the frontier.

6.1.3 Adding the Best Match to the Brokenness Group

Now a new match from the frontier should be added to the current Brokenness Group. So we iterate through the frontier matches. For every frontier match it is calculated what the error value of the current Brokenness Group would be, if it were added to the Brokenness Group. This error is the Mean Squared Error (MSE) of the best fitting between the vertex locations of the vertices in the set from G and the matched vertices in the set from G'. It is calculated using Horn's algorithm.

At the end the best match, i.e., the one which leads to the lowest overall MSE, is taken. If this error value is above the threshold $\epsilon_{brokenThres}$ we return with the current Brokenness Group. Otherwise the best match is added to the Brokenness Group and its neighbors are added to the frontier using the approach described above. This process of calculating the error values of the frontier and adding the best match is repeated until either the frontier is empty or $\epsilon_{brokenThres}$ has been exceeded.

6.1.4 Finding all Brokenness Groups

So far it was described how to grow one Brokenness Group using a Wavefront Propagation approach. Finding all Brokenness Groups is then fairly simple. All matches of vertices between G and G' are iterated. If the match is not already in a Brokenness Group it is taken as the initial match for the Wavefront Propagation. The resulting Brokenness Group that this match created is saved.

After that the biggest Brokenness Group is selected. If it has at least $\epsilon_{brokenMinV}$ vertices it is added to the final set of Brokenness Groups. This process is repeated until no Brokenness Group can be extracted anymore.

The reason why the number of vertices is taken as a selection criterion and not the error value is, that the error value will always tend towards the threshold. Matches are added until there is none left that would not exceed the threshold. So the values of the errors are often similar. Taking the biggest group is a good choice, since one can be fairly certain that a big group within the error threshold is actually a good matching map part.

6.1.5 Handling Unmatched Vertices

It can happen that a part of a map that is broken is not represented by a fully connected Topology Graph. This happens if some vertices are not matched, i.e., no correspondences for them were found, e.g., because the local grid map information is severely erroneous. In this case there are two Brokenness Groups generated, because the Wavefront Propagation cannot "skip" through unmatched vertices. Following approach mitigates this effect.

After finding a Brokenness Group of at least the minimum size, all unused matches between vertices from G and G' tried to be added to the brokenness group, i.e., they are processed as described in Section 6.1.3 for the good matches: the one with the lowest MSE is selected and added if the threshold $\epsilon_{brokenThres}$ is not exceeded. This is repeated until no more unused matches are left or all of the vertices exceed the threshold.

6.2 Brokenness Experiment

The six maps that were used in [6] are also used in the following experiment for comparison. Map 0 (Figure 13) serves as the ground map while Maps 1 to 5 (Figures 13 and 14) are the maps that are evaluated. The computation is done with $\epsilon_{brokenThres} = 80$ as threshold for the mean squared error and $\epsilon_{brokenMinV} = 5$ for the minimum number of vertices in an accepted group.

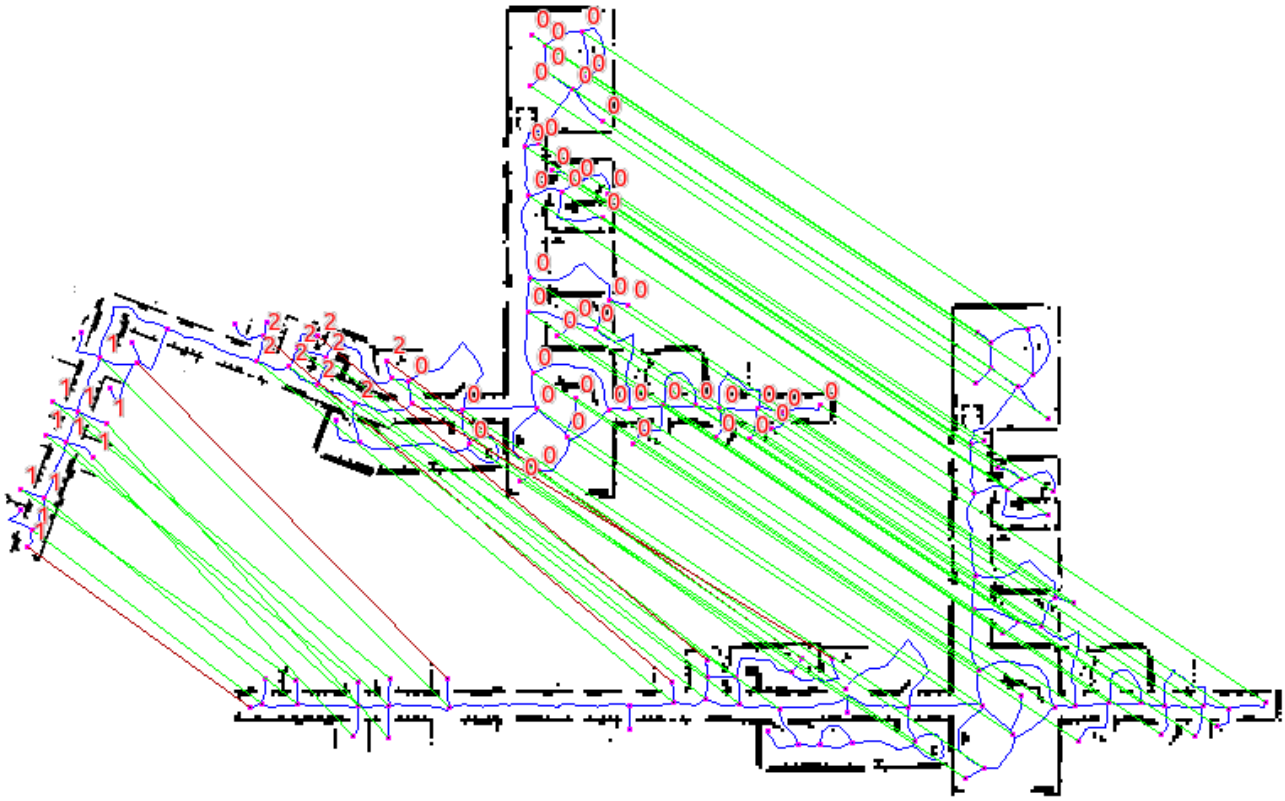


Fig. 13 The Ground Truth Map (Map 0 - bottom) matched to Map 2 (top), which has two broken areas. The green matches are made by the Topology Graph Isomorphism and the additional red matches by a subsequent run of the Neighbor Growing algorithm. The numbers represent the brokenness group each vertex belongs to.

Map	Number of		Matching Vertices with Ground Truth		
	Vert.	E.	Isomorphism	Neighbor Growing	Missing
0	75	88	-	-	-
1	74	86	58	8	8
2	76	89	61	6	8
3	79	94	49	9	17
4	79	93	50	9	16
5	79	93	47	9	19

Table 2 Data of the Topology Graphs of the different maps of the brokenness experiment and their matches against the ground truth map. (Vert. = Vertices; E. = Edges)

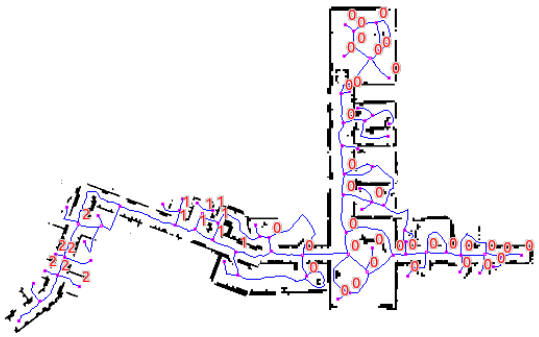
In Figure 13 Map 2 is shown together with the matches to the ground truth map and the Brokenness Groups found. The numbers represent the Brokenness Group the vertex belongs to. The biggest group with 44 matches (which also means 44 vertices per map) is number 0 - this part represents the unbroken map. Horn's algorithm of course also calculates the transformation between two point sets. The angle from this transformation is 0 degrees for set 0 and 20.6 degrees for set 1, which is correct.

Map	Brok.	Set Number	# Matches	Squared Error	Angle in °
1	1	0	44	20.8	0.0
		1	18	57.3	20.6
2	2	0	44	58.5	0.1
		1	11	8.9	-69.5
		2	9	79.9	21.8
3	2	0	35	68.5	0.1
		1	8	69.9	23.4
		2	6	44.4	-70.8
4	3	0	28	77.7	0.3
		1	10	79.7	7.4
		2	9	79.9	21.8
		3	7	77.3	-70.5
5	4	0	19	76.8	0.7
		1	9	71.4	7.4
		2	9	79.9	21.8
		3	7	77.3	-70.5
		4	6	53.8	7.8

Table 3 The different brokenness groups. (Brok. = Detected Brokenness)

Table 2 shows some statistics for the graphs of the maps and their matches. In Table 3 the different Brokenness Groups for the matches are presented. Please

(a) Map 1: Brokeness degree 1



(b) Map 3: Brokeness degree 3

(c) Map 4: Brokeness degree 4

(d) Map 5: Brokeness degree 5

Fig. 14 The brokenness groups for the different maps.

Map	Coverage	Relative Accuracy	Consistency		
			Short	Medium	Long
1	0.88	0.71	0.95	0.88	0.41
2	0.89	0.62	0.94	0.84	0.29
3	0.77	0.63	0.87	0.68	0.25
4	0.79	0.63	0.86	0.70	0.17
5	0.75	0.59	0.82	0.53	0.13

Table 4 The results of the Topology Map Evaluation.

also refer to Figures 13 and 14 for a visualization of these results. The evaluation of maps 1 and 2 generates the expected result, i.e., a brokenness of 1, respectively 2 is detected. For maps 3, 4 and 5, the left-most brokenness cannot be detected. It is a very small area that is affected and it is only represented by three vertices. Since the minimum number of vertices in a brokenness group ($\epsilon_{brokenMinV}$) is configured with 5 this broken part cannot form its own set. All other broken parts from maps 3, 4 and five are properly detected.

In Table 4 the results for the other map attributes are shown. The inherent ranking of the maps (maps with higher numbers being more often broken and thus worse than lower number maps) is nicely reflected in the Consistency attributes. Due to the brokenness some vertices cannot be matched and thus the coverage is not 100% for the maps. But still the value is and remains quite high for the different maps, reflecting that they represent a fair amount of the actual environment.

One can see that the Relative Accuracy does also not change too much with the level of brokenness. This is because sometimes the next brokenness is bend towards the "correct" direction, thus potentially even improving the Relative Accuracy. As expected, the Consistency attributes are the strongest indicator that is corralated with the brokenness. Three Local Consistency attributes are calculated: One for graph-distances between two vertices from the ground truth graph (Map 0) of less than 60 (short), one for distances between 60 and 180 (medium) and one for higher distances (long). The medium and long range consistency values significantly decrease in value with every brokenness that is added to the map. Since the broken parts do not have any other errors, the short range consistency is quite high and only slowly decreasing due to the errors around the start areas of the broken parts.

The runtime for the experiments are very short. The extraction of the Topology Graphs takes much longer than the finding of the Brokenness Groups. Everything together is done for one of the experiment maps in under one second. The algorithm finds the brokenness almost as reliable as [6]. Only very small scale broken parts which are not represented by enough vertices in the Topology Graph cannot be detected. But the im-

age registration approach used in [6] relies on similar "looking" maps, i.e., distinct local features of the grid map - and it has a much higher runtime.

6.3 Experiments with the Interleague Mapping Challenge 2010 Maps

The following experiments are performed on maps generated during the Interleague Mapping Challenge 2010 [21]. The challenge was designed to compare the performance of mapping algorithms. All other factors that influence the quality of maps such as sensors, the environment, the path taken or the processing power, are the same for all contestants due to the special setup of the competition. This is achieved by the organizers by providing the sensor data and the computation environment.

The sensor data was gathered during the RoboCup Rescue Competition 2010 in Singapore, while the training data was collected at the robot test facilities at the National Institute of Standards and Technology (NIST) in Gaithersburg, Maryland, earlier that year. The data consists of the readings of a Laser Range Finder (LRF) and an Inertial Measurement Unit (IMU). Some datasets were used twice, with the second set altered in two specific ways. The first change was, in order to simulate a LRF with shorter range, that the range data was capped at 5 meters. Second, the order in which the data was presented to the algorithm could be reversed.

The sensor data was provided to the mapping algorithm in a defined, binary format provided by the University of Koblenz [21]. The sensors used for the data collection of the Interleague Mapping Challenge are a Hokuyo UTM-30LX laser range finder (LRF) with a field of view of 270°, an angular resolution of 0.25° and a range of slightly above 30m as well as a Xsens MTi gyro and accelerometer. Those were mounted on a stick and connected to a Laptop. The sensor data was collected by a person, holding the stick with the sensors, slowly walking through the maze and the environment.

Eight maps representing the final round of the mapping competition are evaluated against the ground truth map, shown in Figure 15. The ground truth data of the environment used for the mapping finals was collected in the maze configuration of the last day of RoboCup 2010 in Singapore, in which the runs for the autonomy- and the mapping-awards were conducted. Table 5 lists the datasets used. The ground truth map from Figure 15 was generated by using nearly perfect sensor data (a lightweight robot was carried through the maze) and the path included several loops, such that a very good map could be created.

Maps	Map Name	Description
0	Ground truth	No dataset, but the ground truth map from Figure 15.
I & V	mappingFinal	The original dataset including LRF (up to 30m range) and IMU data.
II & VI	mappingFinal-5m	The above dataset with LRF range capped at 5 meter.
III & VII	mappingFinal-backwards	The original dataset in reverse replaying order.
IV & VIII	mappingFinal-backwards-5m	The above, reversed dataset capped at 5 meter.

Table 5 The "mapping final" datasets from the Interleague Mapping Challenge 2010.

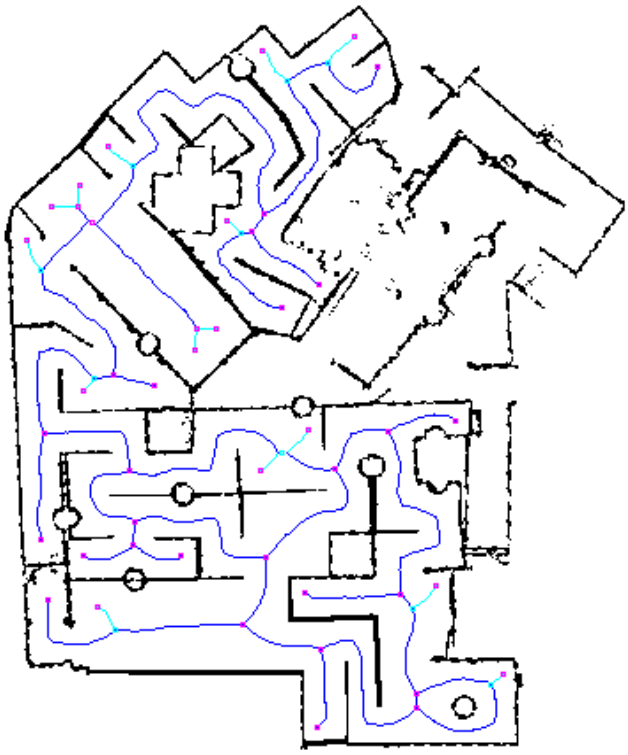


Fig. 15 The ground truth map.

Two mapping algorithms, one provided by the University of New South Wales, Australia, and one from the University of Koblenz, Germany were tested. Figures 16 and 17 show the resulting maps and how they were matched against the ground truth map. No global reference was used; so the relative global position of the maps varies. Thus the general direction of the lines representing the matches with the ground truth topology graph, which is not shown, varies for each map.

For Map 1 the brokenness algorithm detected a brokenness degree of one with an associated angle of 7.6 degrees. The brokenness groups for this map are shown, but omitted for all other maps, since for those no bro-

kenness was detected. The broken parts in the map are indicated with arrows.

6.3.1 Results

Table 6 shows the results of the experiments. The graph of Map VIII differs too much from the ground truth graph, such that it could not be matched. This is because of blocked or too narrow passages in the map and other mapping errors that can be seen when carefully comparing this map to the ground truth map. The ability to sense such circumstances although the map does not look too bad on first glance is an important aspect of this map evaluation algorithm.

Three Local Consistency attributes have been calculated (see [25] for a detailed definition): One for distances between two vertices of a graph of less than 6 meters (short), another for distances between 6 and 18 meters (medium) and one for higher distances (long).

The map attributes have also been combined into a weighted result. The coverage was given the most importance, because it is the only attribute that penalizes missing matches. The other attributes work on the matched vertices and could thus return good values even if half of the map has really bad quality. So for the weighted result, the Relative Accuracy and the Consistencies are averaged in one value which is in turn averaged with the Coverage attribute.

In Table 7, the rankings among the maps that are derived from the different attributes are shown. A qualitative ranking made by the authors is included as a basis of comparison for the map metric. This qualitative human ranking was done a priori to the quantitative evaluations and it is motivated as follows:

Maps VI and V are very good and reflect the geometry of the environment better than Map I. Maps III and IV as well as VII and VIII have problems in the lower half of the map. But Maps III and IV are still better than VII and VIII - the paths to a free standing barrel in the lower right corner is correctly shown, which is not the case for maps VII and VIII. Map IV is better than Map III, because of the better represented dead end on the left and also because of the top parts, that are shown more accurately. Map II contains severe errors in the right side (e.g. location of a free standing barrel) and it is hence put into the worst class together with Maps VII and VIII.

The rankings from Table 7 show that the Topology Map Metric corresponds quite well with the human judgment of the maps. Furthermore, brokenness could be detected in the upper part of Map I. This is a very nice result, because this fact could have been

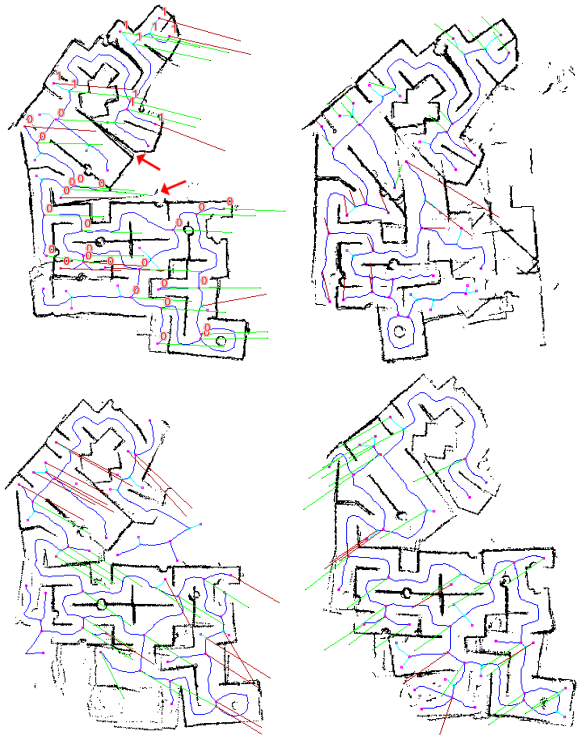


Fig. 16 Maps I (top left), II (top right), III (bottom left) and IV (bottom right). The broken parts of Map I are indicated with arrows.

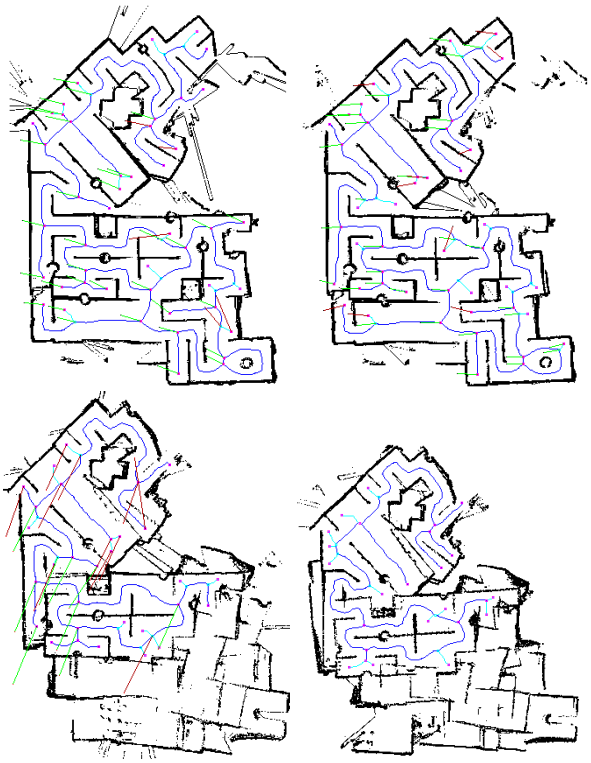


Fig. 17 Maps V (top left), VI (top right), VII (bottom left) and VIII (bottom right).

Map	Number Vertices	Number Matches	Number Major Matches	Coverage	Relative Accuracy	Consistency			Weighted Result
						Short	Med.	Long	
I	57	36	30	0.68	0.89	0.83	0.81	0.45	0.71
II	53	29	23	0.52	0.83	0.77	0.58	0.47	0.59
III	62	32	26	0.59	0.88	0.78	0.59	0.30	0.61
IV	55	34	28	0.64	0.95	0.85	0.80	0.76	0.74
V	55	39	34	0.77	0.96	0.85	0.81	0.82	0.81
VI	59	43	35	0.80	0.95	0.89	0.77	0.78	0.82
VII	38	20	16	0.36	0.87	0.79	0.73	0.44	0.53
VIII	31	0	0	0	0	0	0	0	0

Table 6 The results of the Topology Map Evaluation for the Interleague Mapping Challenge maps.

Map	H.	W.	C.	Rel. Acc.	Consistency		
					Short	Med.	Long
V	1-2	2	2	1	2-3	1-2	1
VI	1-2	1	1	2-3	1	4	2
I	3	4	3	4	4	1-2	5
IV	4	3	4	2-3	2-3	3	3
III	5	5	5	5	6	6	7
II	6-8	6	6	7	7	7	4
VII	6-8	7	7	6	5	5	6
VIII	6-8	8	8	8	8	8	8

Table 7 The rankings of the Topology Map Evaluation for the Interleague Mapping Challenge maps, sorted by the human ranking. H.:Human Ranking, W.: Weighted Ranking, C.: Coverage, Rel. Acc.: Relative Accuracy.

easily missed by a human judge. The brokenness in the map can be seen when looking at the V-shaped walls indicated by the red arrows. Although maps II, VII and VIII contain large errors in the bottom parts, those areas don't qualify as broken, because they also contain in itself lots of errors.

7 Conclusions

A novel approach for map evaluation using Topology Graphs has been introduced in this article. The main advantage of this approach is that it abstracts from the grid representation with the occupied cells and employs the underlying topology of the maps. After matching two Topology Graphs, standard map quality attributes like coverage, global accuracy, relative accuracy, consistency, or brokenness can then be efficiently computed in a reliable manner.

The computation of a Topology Graph is based on the well-known Voronoi Diagram. Starting from a raw Voronoi Diagram extracted from an occupancy grid map, several processing steps are used to prune and simplify the graph such that only a much smaller amount of high level topological environment information remains, i.e., that especially junctions and larger places are represented by single interconnected vertices. The Topology Graph also maintains local metric information like the relative angles in which edges - or more precisely, the related metric paths - leave a vertex. This abstracted metric information can be used for place-recognition.

Several methods for computing the similarity of vertices in two Topology Graphs, i.e., for performing a place-recognition, were presented. The methods fall into two different categories, namely topological ones using information from the Topology Graphs including the abstracted local metric information and sensor-based ones that use the grid-map data in the direct neighborhood of the vertices. It was shown in an experimental evaluation that topological methods can be quite robust and that they are computationally extremely efficient. Furthermore, it is shown that the robustness can be significantly boosted by exploiting the neighborhood relations in the Topology Graphs: if two vertices are similar then their neighbors should also be similar if the two vertices indeed correspond to the same place. It was shown through the strategy of propagation rounds that this principle can be very efficiently applied to vertex similarity computations.

Vertex similarity allows for an initial assignment of possible correspondences between the vertices of two Topology Graphs, i.e., to identify the same places in two different maps. Based on this, a strategy for computing (sub)graph-isomorphisms was introduced to find large connected components in the graphs that can be matched to each over. The method uses a wavefront propagation in combination with a few heuristics to be computationally very efficient.

Finally, it was shown in several experiments that the Topology Graphs approach is indeed beneficial for map evaluation with standard map quality attributes like coverage, global accuracy, relative accuracy, consistency, and brokenness. While this is out of the scope of this article, it is interesting to note that the Topology Graph with its combination of topological and abstracted local metric information can also be used for applications beyond map evaluation. The topological computation of Vertex Similarity can for example be applied to place-recognition in general and the Graph

Matching may for example be used for map merging and cooperative multi-robot mapping.

Acknowledgements The authors would like to thank all RoboCup Rescue teams that agreed to have their maps published in the context of this research: CASualty: University of New South Wales, Australia and Resko: University of Koblenz, Germany.

References

1. Aurenhammer, F.: Voronoi diagramsã survey of a fundamental geometric data structure. ACM Comput. Surv. **23**, 345–405 (1991). DOI http://doi.acm.org/10.1145/116873.116880. URL http://doi.acm.org/10.1145/116873.116880
2. Bai, X., Latecki, L.: Path similarity skeleton graph matching. Pattern Analysis and Machine Intelligence, IEEE Transactions on **30**(7), 1282–1292 (2008). DOI 10.1109/TPAMI.2007.70769
3. Bai, X., Latecki, L., yu Liu, W.: Skeleton pruning by contour partitioning with discrete curve evolution. Pattern Analysis and Machine Intelligence, IEEE Transactions on **29**(3), 449–462 (2007). DOI 10.1109/TPAMI.2007.59
4. Balaguer, B., Balakirsky, S., Carpin, S., Visser, A.: Evaluating maps produced by urban search and rescue robots: lessons learned from robocup. Autonomous Robots **27**, 449–464 (2009)
5. Besl, P.J., McKay, N.D.: A method for registration of 3-d shapes. IEEE Trans. on Pattern Analysis and Machine Intelligence **14**(2), 239–256 (1992)
6. Birk, A.: A quantitative assessment of structural errors in grid maps. Autonomous Robots **28**, 187–196 (2010)
7. Birk, A., Wiggerich, B., Bülow, H., Pfingsthorn, M., Schwertfeger, S.: Safety, security, and rescue missions with an unmanned aerial vehicle (uav): Aerial mosaicking and autonomous flight at the 2009 european land robots trials (elrob) and the 2010 response robot evaluation exercises (rree). Journal of Intelligent and Robotic Systems **64**(1), 57–76 (2011)
8. Chen, Q.S., Defrise, M., Deconinck, F.: Symmetric phase-only matched filtering of fourier-mellin transforms for image registration and recognition. Pattern Analysis and Machine Intelligence, IEEE Transactions on **16**(12), 1156–1168 (1994)
9. Edelsbrunner, H., Kirkpatrick, D., Seidel, R.: On the shape of a set of points in the plane. Information Theory, IEEE Transactions on **29**(4), 551 – 559 (1983). DOI 10.1109/TIT.1983.1056714
10. Fortin, S.: The graph isomorphism problem. Tech. rep. (1996)
11. Handa, A., Whelan, T., McDonald, J., Davison, A.: A benchmark for rgb-d visual odometry, 3d reconstruction and slam. In: Robotics and Automation (ICRA), 2014 IEEE International Conference on, pp. 1524–1531 (2014). DOI 10.1109/ICRA.2014.6907054
12. Horn, B.K.P.: Closed-form solution of absolute orientation using unit quaternions. Journal of the Optical Society of America **4**(4), 629–642 (1987)
13. Karavelas, M.: 2D Voronoi diagram adaptor. In: CGAL User and Reference Manual, 3.8 edn. CGAL Editorial Board (2011). http://www.cgal.org/Manual/3.8/doc_html/cgal_manual/packages.html#Pkg:VoronoiDiagramAdaptor2
14. Klein, R.: Abstract voronoi diagrams and their applications. In: H. Noltemeier (ed.) Computational Geometry and its Applications, *Lecture Notes in Computer Science*, vol. 333, pp. 148–157. Springer Berlin / Heidelberg (1988)
15. Kümmerle, R., Steder, B., Dornhege, C., Ruhnke, M., Grisetti, G., Stachniss, C., Kleiner, A.: On measuring the accuracy of slam algorithms. Autonomous Robots **27**(4), 387–407 (2009)
16. Lakaemper, R., Adluru, N.: Using virtual scans for improved mapping and evaluation. Auton. Robots **27**(4), 431–448 (2009). DOI http://dx.doi.org/10.1007/s10514-009-9149-4
17. Lee, D.C.: The Map-Building and Exploration Strategies of a Simple Sonar-Equipped Mobile Robot: An Experimental, Quantitative Evaluation. Distinguished Dissertations in Computer Science. Cambridge University Press (1996)
18. Leonard, J.J., Durrant-Whyte, H.F.: Directed Sonar Sensing for Mobile Robot Navigation. Springer (1992)
19. Pellenz, J., Paulus, D.: Mapping and Map Scoring at the RoboCupRescue Competition. Quantitative Performance Evaluation of Navigation Solutions for Mobile Robots (RSS 2008, Workshop CD) (2008)
20. Pomerleau, F., Liu, M., Colas, F., Siegwart, R.: Challenging data sets for point cloud registration algorithms. The international Journal of Robotics Research **31**(14), 1705–1711 (2012)
21. Schwertfeger, S.: Robocuprescue interleague mapping challenge (2010). http://robotics.jacobs-university.de/mappingChallenge/
22. Schwertfeger, S.: Robotic mapping in the real world: Performance evaluation and system integration. Ph.D. thesis, Department of Electrical Engeneering and Computer Science, Jacobs University Bremen (2012). URL http://www.jacobs-university.de/phd/files/phd20120706_Schwertfeger.pdf
23. Schwertfeger, S., Birk, A.: Evaluation of map quality by matching and scoring high-level, topological map structures. In: Robotics and Automation (ICRA), 2013 IEEE International Conference on (2013)
24. Schwertfeger, S., Bülow, H., Birk, A.: On the effects of Sampling Resolution in Improved Fourier Mellin based Registration for Underwater Mapping. In: 7th International Symposium on Intelligent Autonomous Vehicles (IAV 2010). IFAC (2010)
25. Schwertfeger, S., Jacoff, A., Pellenz, J., Birk, A.: Using a fiducial map metric for assessing map quality in the context of robocup rescue. In: International Workshop on Safety, Security, and Rescue Robotics (SSRR). IEEE Press (2011)
26. Schwertfeger, S., Jacoff, A., Scrapper, C., Pellenz, J., Kleiner, A.: Evaluation of maps using fixed shapes: The fiducial map metric. In: Proceedings of PerMIS (2010)
27. Scrapper, C., Madhavan, R., Balakirsky, S.: Stable navigation solutions for robots in complex environments. In: IEEE International Workshop on Safety, Security and Rescue Robotics (SSRR), pp. 1–6 (2007)
28. Siddiqi, K., Shokoufandeh, A., Dickenson, S., Zucker, S.: Shock graphs and shape matching. In: Computer Vision, 1998. Sixth International Conference on, pp. 222–229 (1998). DOI 10.1109/ICCV.1998.710722
29. Sturm, J., Engelhard, N., Endres, F., Burgard, W., Cremers, D.: A benchmark for the evaluation of rgb-d slam systems. In: Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, pp. 573–580 (2012). DOI 10.1109/IROS.2012.6385773

30. Texas A&M University College Station, T.: Nist response robot evaluation exercise (2008). URL http://www.teex.com/teex.cfm?pageid=USARprog&area=usar&templateid=1538
31. Thrun, S.: Robotic mapping: A survey. In: G. Lakemeyer, B. Nebel (eds.) Exploring Artificial Intelligence in the New Millenium. Morgan Kaufmann (2002)
32. Torsello, A., Hancock, E.: A skeletal measure of 2d shape similarity. In: C. Arcelli, L. Cordella, G. di Baja (eds.) Visual Form 2001, *Lecture Notes in Computer Science*, vol. 2059, pp. 260–271. Springer Berlin Heidelberg (2001). DOI 10.1007/3-540-45129-3_23
33. Varsadan, I., Birk, A., Pfingsthorn, M.: Determining map quality through an image similarity metric. In: L. Iocchi, H. Matsubara, A. Weitzenfeld, C. Zhou (eds.) RoboCup 2008: Robot WorldCup XII, Lecture Notes in Artificial Intelligence (LNAI), pp. 355–365. Springer (2009)
34. Voronoi, G.: Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Premier mémoire. Sur quelques propriétés des formes quadratiques positives parfaites. Journal für die reine und angewandte Mathematik **133**, 97–102 (1908). DOI 10.1515/crll.1908.133.97. URL http://dx.doi.org/10.1515/crll.1908.133.97
35. Wagan, A.I., Godil, A., Li, X.: Map quality assessment. In: PerMIS '08: Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems, pp. 278–282. ACM, New York, NY, USA (2008). DOI http://doi.acm.org/10.1145/1774674.1774718
36. Wulf, O., Nüchter, A., Hertzberg, J., Wagner, B.: Ground truth evaluation of large urban 6d slam. pp. 650 –657 (2007). DOI 10.1109/IROS.2007.4399026
37. Zemlyachenko, V.N., Korneenko, N.M., Tyshkevich, R.I.: Graph isomorphism problem. Journal of Mathematical Sciences **29**(4), 1426–1481 (1985). DOI 10.1007/BF02104746. URL http://dx.doi.org/10.1007/BF02104746