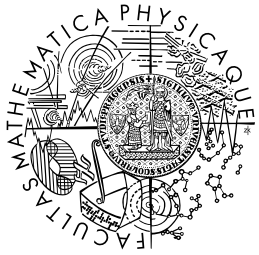


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Jiří Hörner

Automatic Point Clouds Merging

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis:	RNDr. David Obdržálek, Ph.D.
Study programme:	Computer Science
Study branch:	Artificial Intelligence

Prague 2018

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

I wish to thank to Univ.Prof. Dr. Stephan Weiss and Dejan Sazdov from Alpen-Adria-Universität Klagenfurt for kindly providing data for one of the datasets. I wish to thank to my family for their support.

Title: Automatic Point Clouds Merging

Author: Jiří Hörner

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: RNDr. David Obdržálek, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: Multi-robot systems are an established research area with a growing number of applications. Efficient coordination in such systems usually requires knowledge of robot positions and the global map. This work presents a novel map-merging algorithm for merging 3D point cloud maps in multi-robot systems, which produces the global map and estimates robot positions. The algorithm is based on feature-matching transformation estimation with a novel descriptor matching scheme and works solely on point cloud maps without any additional auxiliary information. The algorithm can work with different SLAM approaches and sensor types and it is applicable in heterogeneous multi-robot systems. The map-merging algorithm has been evaluated on real-world datasets captured by both aerial and ground-based robots with a variety of stereo rig cameras and active RGB-D cameras. It has been evaluated in both indoor and outdoor environments. The proposed algorithm was implemented as a ROS package and it is currently distributed in the ROS distribution. To the best of my knowledge, it is the first ROS package for map-merging of 3D maps.

Keywords: ROS map-merging Point cloud multi-robot systems

Contents

Introduction

Multi-robot systems are established research area within robotics and artificial intelligence with a growing number of applications. A multi-robot system, consisting of several individual robots, can improve efficiency in terms of both performance and robustness over a single robot. A team of robots can also solve tasks that cannot be tackled by a single robot. Multi-robot systems can scale to large environments and to the most demanding applications.

Multi-robot systems have been deployed to several application domains including space exploration [15, 22], autonomous patrolling [35], disaster rescue and victim search, aerial surveillance, unmanned delivery, mine cleaning, snow removal [8], assembly of large-scale buildings or planetary habitat [15] and robotic football [2]. Those applications require robots to operate in dynamic environments, with a high degree of uncertainty and external changes caused by other robots, humans and other agents that are not part of the multi-robot system itself.

To be able to solve such demanding problems multi-robot systems rely on distributed planning, communication and control algorithms. This work presents an algorithm for estimating positions of the robots in the shared environment and for building the global map of the environment (map-merging) without any previous knowledge or assumptions about the environment. Knowledge of positions of the robots and the map of the environment are crucial elements of effective collaborative planning and coordination. For most of the multi-robot systems, this knowledge is essential and required for the operation, making the map-merging a key problem to solve. Multi-robot systems that don't have knowledge of the presence of other robots (*unaware systems* [10]) are normally used only for very simple tasks, as noted by Farinelli et al. [10].

A typical multi-robot system consists of many software parts typically structured in a multi-layer architecture. The implementation presented in this work leverages modularity of the widely-used Robot Operating System (ROS) framework and respects community established standards. This allows easy integration with existing planning, mapping and communication algorithms enabling quick development of multi-robot systems suited for the particular task. To my best knowledge, the presented implementation is the first implementation of a three-dimensional (3D) map-merging algorithm for multi-robot systems within the ROS ecosystem.

Chapter 1 of this work discusses related works and various approaches to estimation of robot positions in an unknown environment. Chapter 2 introduces the presented map-merging algorithm. Chapter 3 presents the implementation of the developed map-merging algorithm for the ROS framework. Chapter 4 accesses the performance of the implementation using several standard robotic datasets and experiments done by the author. Documentation for the ROS implementation is attached as Appendix A.

1. Analysis

Knowledge of the positions of robots and knowledge of the environment is essential for solving complex tasks efficiently in multi-robot systems. Over time various techniques evolved to acquire this knowledge.

1.1 Map-merging

This work focuses on multi-robot systems consisting of mobile robots, each capable of building a 3D map of the surrounding environment. Typically, each robot uses a Simultaneous Localization and Mapping (SLAM) algorithm to create a map and is equipped with appropriate sensors such as stereo camera rigs, active Red-Green-Blue-Depth (RGB-D) cameras or laser rangefinders. Such sensors are available in the broad weight and cost range enabling a large variety of SLAM-capable robots operating both on the ground and in the air. This variety leads to possible heterogeneous teams of robots capable of solving a broad range of tasks.

The *map-merging problem* for multi-robot systems, which is the focus of this work, is to estimate positions of the robots in the environment and to merge the maps from individual robots to produce a single globally consistent map.

1.2 The map representation

3D maps are becoming more common in the robotics as the capable sensors are getting more affordable and suitable for many applications. Unlike two-dimensional (2D) occupancy grids used extensively in the past, 3D maps enable cooperation of robots, that are not fixed to a 2D plane, such as aerial vehicles, humanoid robots, outdoor robots operating in a rough terrain as well as traditional ground-based platforms. This allows us to create possibly heterogeneous multi-robot teams, that are able to take advantage of their different strengths and weaknesses to solve the assigned task efficiently. For example, a robotic team can consist of aerial vehicles, capable of fast reconnaissance in large-scale outdoor environments, and ground-based vehicles carrying heavy equipment.

This work expects maps represented as point clouds (Definition 1). Point clouds can be implemented as an array of points, making them suitable for serialisation and exchange between robots. Points in the point cloud can have assigned additional information, such as Red-Green-Blue (RGB) colour information or reflection intensity, based on the available sensor. Especially colour information, that is provided by stereo camera rigs and active RGB-D cameras is commonly exploited for further processing (Figure 1.1).

Definition 1 (Point cloud)**.** *A point cloud is a set of data points in space.*

Other data structures used in the 3D mapping, such as octree-based maps used by Hornung et al. [21], can be converted to point clouds without any loss of information because point clouds do not pose any restrictions on the geometry of the points and it is easy to add metadata associated with the points. This makes

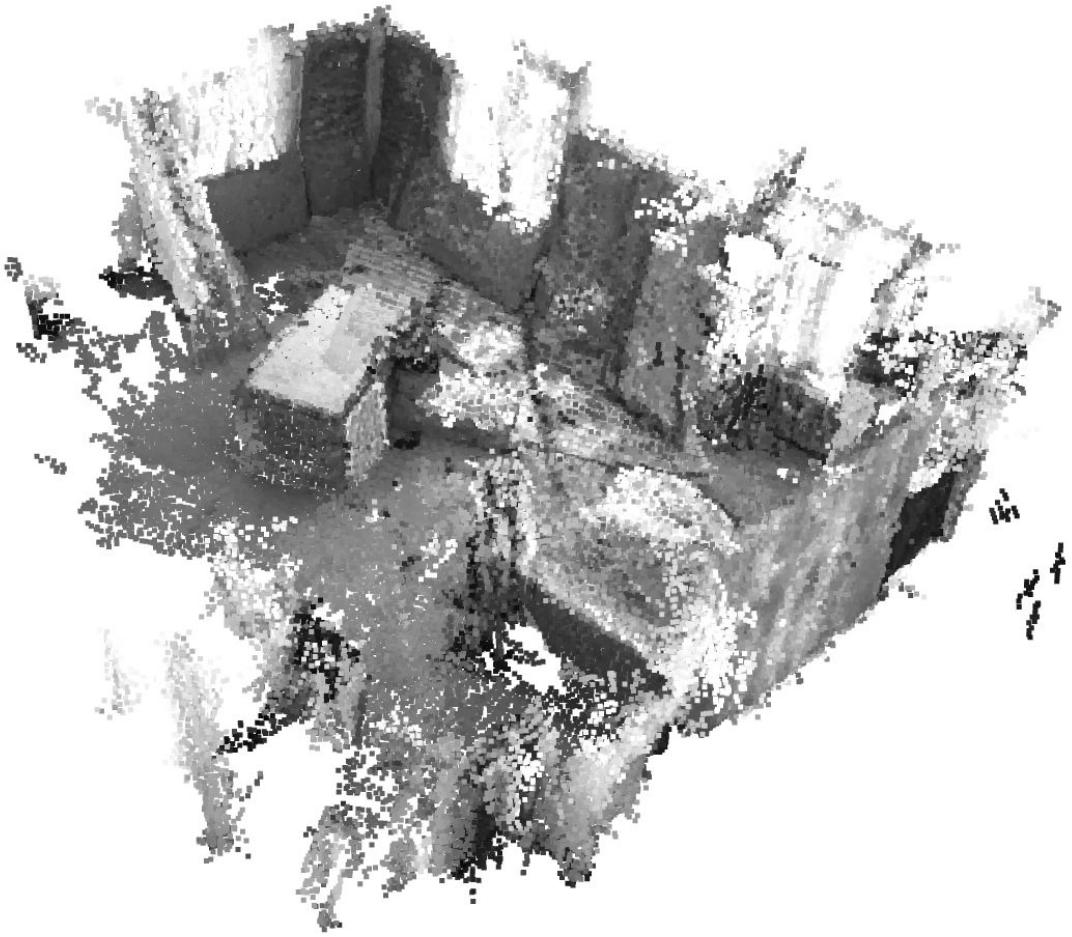


Figure 1.1: Partial point cloud map of a room created from the aerial vehicle equipped with greyscale stereo cameras rig. Intensity captured by the camera is stored alongside the points and visualised in greyscale. The map was created from The EuRoC micro aerial vehicle dataset "Vicon Room 1 02" (Section 4.1).

point clouds a suitable data structure for exchange between different mapping approaches, even when they use different representations internally.

Point clouds data messages are supported and well established in the ROS framework. There are mature libraries to work with point clouds, such as Point Cloud Library (PCL). This makes point clouds suitable map representation for multi-robot systems.

1.3 Related works

The map-merging is often a core part of a multi-robot system and various approaches have been developed to solve the map-merging problem. Most of the listed techniques work with robots operating in the 2D space (ground-based robots), but some of those techniques may be adapted to work in the 3D space too. The robot rendezvous techniques (Section 1.3.1) and *indirect map-merging* techniques (Section 1.3.2) have been used mostly with a 2D assumption, while distributed SLAM techniques (Section 1.3.3) can be used also in the 3D case.

1.3.1 Direct map-merging

Early map-merging techniques, classified as *direct map-merging* by Lee et al. [28], use direct sensor measurements to compute the transformation between robots. This involves especially the case of robot rendezvous used by Zhou and Roumeliotis [44], which relies on robots' ability to sense each other position directly. Popular techniques use a camera and special recognizable markers on the robots to compute the transformation using computer vision algorithms.

A robot rendezvous might be hard to achieve in large-scale environments, especially if the robots are starting from different locations or in a different time (for example to react to increasing demands for the service). Such applications typically need map-merging to be able to work only with the sensed surrounding environment instead of the presence of other robots in the same space.

Direct map-merging also involves techniques relying on global localization in the environment being available. This involves purpose-adapted environments with installed markers or beacons to locate robots within environment and systems using global localization such as Global Positioning System (GPS).

While purpose-adapted environments may greatly simplify the map-merging problem, they make deployment of the multi-robot systems harder and might introduce severe costs. For applications such as space exploration, disaster rescue, victim search and mine cleaning any modifications of the environment are not conceivable and in many more applications such as unmanned delivery, the modifications are usually not practical.

Global localization systems are usually not available indoor and while such systems are a great aid for navigating large-scale outdoor environments, the provided accuracy is usually not high enough for merging high-resolution point cloud maps. The position from global localization systems can be however used as an initial guess for map-merging algorithms.

1.3.2 Map-merging on occupancy grids

Later developed techniques work with 2D maps, usually represented as occupancy grids. Lee et al. [28] categorize these techniques as *indirect map-merging*. These techniques rely on overlapping areas in maps for map-merging.

Using 2D maps limits operational space of robots and practically limits multi-robot teams to only ground-based vehicles operating indoor or in light terrain, flat environments. Particularly 2D maps prohibit promising multi-robot teams consisting of aerial and ground-based vehicles.

Map-merging techniques for 2D occupancy grids involves spectra-based approach of Carpin [7] and my previous work based on image feature approach [23].

My previous work is available in the ROS distribution as a ready-made solution for merging 2D maps. It influenced the presented 3D map-merging algorithm as I have gathered feedback from the ROS community. From experience, it is especially important to select a suitable interoperable map format in ROS, so that the implementation can be used with existing SLAM algorithms in ROS. The idea of automatic robot discovery (Section 3.3.3) enabling convenient configuration and the idea of decoupling inter-robot communication platforms and the map-merging algorithm (Section 3.3.1) are reused in the presented implementation (Chapter 3). However, the presented map-merging algorithm (Chapter 2) has been designed from scratch and is based on different techniques as the data structures for 2D maps (occupancy grid) and 3D maps (point cloud) are different. The idea of converting occupancy grids to images and the idea of reusing image-based techniques, as used in my previous work, are not applicable.

1.3.3 Map-merging as distributed SLAM

Map-merging algorithms have been also implemented as distributed SLAM algorithms. These techniques take advantage of the research in the area of SLAM algorithms and use loop-closure techniques developed for SLAM to implement map-merging. Especially graph-based SLAM implementations can be naturally extended to merge maps from multiple robots. Nodes from other robots can be added to the SLAM graph and connected through loop-closures with existing nodes to form a single map.

When a loop-closure between 2 maps is detected, it can be used to compute the transformation between maps. We need to extend the loop detection to work across map boundaries, not only within a single map as usual in the SLAM, which typically requires exchanging implementation-specific data between robots. This poses several disadvantages. Based on the SLAM loop-closure approach the internal SLAM data may be quite large (Table 1.1), stressing communication bandwidth. Exchanging implementation-specific data usually leads to running the same SLAM algorithm on all robots in the multi-robot system, or the implementations must be adapted to exchange compatible loop-closure data. This is challenging for heterogeneous multi-robot systems when individual robots use different sensors, which is often the case when using both ground-based and aerial robots. For example, if ground-based robots are equipped with accurate heavy 3D laser range finder and aerial vehicles use lightweight stereo rig cameras, the SLAM approaches are typically different and the loop-closure data are inherently incompatible (image features and 3D laser scans). In such scenarios, we need to

use a different approach.

Lee et al. [28] list scan matching loop closure approaches used for map-merging, which are used with 2D SLAM algorithms. 2D distributed SLAM based on scan matching was implemented by Pfingsthorn et al. [36] for the RoboCup Rescue Virtual Robots competition. Fox et al. [13] used a direct exchange of laser range scans and odometry motion information coupled with a particle filter to localize a robot in the second map.

In 3D camera-based SLAM algorithms, visual appearance-based techniques are particularly popular. Tomono [43] used an appearance-based algorithm based on Scale-Invariant Feature Transform (SIFT) features to merge visual maps, validated by travelled paths to reject false similarities in repetitive environments. Visual bag-of-words loop closure approach was used by Labbé and Michaud [27] to merge maps in multi-session SLAM. Multi-session SLAM works with maps generated by a single robot starting consecutively in the different positions. In such setup there is no need for communication, so a local database was used by the authors to store loop-closure related data for map-merging.

1.4 Map-merging on point clouds

The algorithm presented in Chapter 2 works exclusively with 3D point clouds without any additional exchange of data to offer maximum flexibility. The algorithm does not require any exchange of implementation-specific data and can be used with any SLAM approach available, allowing future integration of the newest state-of-the-art methods in fast-paced SLAM research. With the presented approach, each robot in the heterogeneous multi-robot system can use a different SLAM implementation, which allows using best-suited implementation to robot sensors and computational capabilities.

Using point clouds can also save communication bandwidth especially compared to image-based loop closure methods, which may require to exchange a significant amount of data. Size comparison between local database size used in multi-session SLAM by Labbé and Michaud [27], implemented in popular `RTAB-Map` SLAM, and point cloud representation of the same maps are shown in Table 1.1. The maps have been recorded at Charles University campus, see Section 4.3 for details. The point clouds are significantly smaller with the resolution of 0.05 meters per voxel. The presented merging algorithm can work with the resolution of just 0.1 meters per voxel in which point clouds are even smaller, so we could save more bandwidth if necessary.

Note that the local database has been used only in multi-session SLAM and might not be size-optimised. Still, the table demonstrates the order of magnitude differences between the two representations.

There are some challenges for map-merging on point clouds. When using SLAM-based approaches for map-merging, the resulting graph is usually optimized, which can repair mapping errors present in the individual maps. Repairing mapping errors in point clouds maps is much harder. Bonanni et al. [4] addressed this problem by using pose graphs of point clouds. In the pose graph representation, the map is represented as a graph of smaller point cloud sub-maps, not as one big point cloud. This representation allows repairing maps in the similar manner as done in distributed SLAM approaches, by optimizing the graph after

merging.

Another challenge is to register the point clouds (to compute the transformation between point clouds) using only information stored within point clouds. The algorithm implemented by Bonanni et al. [4] relies on an external place recognition routine that reports matching pairs of local maps. The external routine is needed because authors used a pairwise registration algorithm, that needs an initialisation with a roughly-correct transformation to align point clouds. The algorithm presented in Chapter 2 can register point clouds directly using a global registration approach that does not need any external place recognition routine. To the best of my knowledge, this approach is the first map-merging method that can work directly on 3D point clouds without any external pre-alignment routine.

Most of the methods for point cloud registration are based on the Iterative Closest Point (ICP) algorithm introduced by Besl and McKay [3]. The ICP algorithm uses an initial transformation estimate and it iteratively tries to minimise error metric (usually a Euclidean distance between point clouds). The initialisation is extremely important as the algorithm always finds a local minimum. Since the introduction of the ICP algorithm, variants, reviewed in Pomerleau et al. [37], of the original algorithm has been developed, which are less susceptible to an accurate initialisation, but all ICP-derived algorithms need the initialisation and are susceptible to local minima problems.

When using ICP-family algorithms for scan matching in the context of SLAM, the initial estimate is usually provided by the odometry source. In the map-merging problem, the initial estimate must be provided from an external source, either manually by the operator or using an automated routine (for example vision-based place recognition method). The need for an accurate initial estimate is a severe drawback in the context of map-merging.

To be able to work without any initial estimate the algorithm presented in Chapter 2 uses a feature matching approach, that does not require any initial estimate and is not affected by the initial configuration of the maps. These feature-matching approaches use descriptors developed specifically for point clouds, such as Point Feature Histogram (PFH), introduced by Rusu et al. [39], to match keypoints between two maps. These algorithms have been developed for high-density point clouds produced directly by sensors such as laser range finder and RGB-D cameras. Some of the proposed features, such as Normal Aligned Radial Feature (NARF) introduced by Steder et al. [41], use range image representation and a single viewpoint assumption, that cannot be generalised to 3D point cloud maps.

Map	**Database**	**Point cloud**	**Ratio**
“MFF Refectory 1”	217 MiB	14 MiB	0.066
“MFF Refectory 2”	551 MiB	33 MiB	0.060
“MFF Rotunda 1”	231 MiB	11 MiB	0.048
“MFF Rotunda 2”	350 MiB	16 MiB	0.046

Table 1.1: Table comparing sizes of the local database of loop closure data for map-merging used by Labbé and Michaud [27] and the point cloud representation of the same maps stored in `pcd` files. Ratio denotes the fraction of the point cloud size to the database size. The maps are from the MFF dataset (Section 4.3).

I show that it is viable to use feature-matching also for registration of low-density point cloud when we take specifics of the point cloud maps into account.

2. The map-merging algorithm

This section presents a novel map-merging algorithm for estimating transformations between n maps and merging them together. The algorithm is based on a feature-matching approach and works solely on 3D point cloud maps without any additional auxiliary information. As discussed in Chapter 1, we work with maps represented as point clouds, possibly with RGB information for each point. To the best of my knowledge, the presented approach is the first map-merging algorithm working directly on point clouds without any extra information.

The feature-matching approach uses point cloud features designed to work with high-density direct sensor measurements, using the features with low-density point cloud maps is challenging. To overcome some of the challenges, I introduce a novel feature matching scheme (Section 2.1.6).

Generally, there are two core problems for estimating the transformations between maps. First, we need to be able to estimate pair-wise transformation for two maps using only geometrical and possibly colour information available within point clouds. We discuss our method in Section 2.1. Second, we want to get a transformation for each of the maps to the selected reference frame. This is discussed in Section 2.2.

After we have estimated the transformations, we can stitch them to create the global map.

2.1 Estimating pair-wise transformation

Algorithm 1 describes an algorithm pipeline to estimate the pair-wise transformation. The algorithm first down-samples the point cloud (Section 2.1.1), then it filters the outliers (Section 2.1.2), after filtering, surface normals are estimated for the whole point cloud (Section 2.1.3). To reduce the number of points for further processing, keypoints are detected using either SIFT-based or Harris keypoint detector (Section 2.1.4). To match keypoints between point clouds, each keypoint is assigned a descriptor (Section 2.1.5). Using assigned descriptors, keypoints are then matched and initial rigid transformation between point clouds is estimated (Section 2.1.6). After matching, the transformation is refined using all points in the filtered point cloud with local minimisation method (Section 2.1.7).

To deal with inaccurate estimates each transformation estimate is evaluated using a confidence measure (Section 2.1.8).

2.1.1 Down-sampling

As we are working with possibly large-scale maps, input point clouds may contain millions of points. To reduce computation times it is highly desirable to reduce the number of points.

A common technique for reducing the resolution of point clouds is the voxelization, which produces a voxel grid. The voxel grid is a regularly spaced, three-dimensional grid (Figure 2.1). We can represent the voxel grid as a normal point cloud, with each point representing a voxel of the voxel grid. We don't usually save empty space information, so the grid is sparse.

Algorithm 1 Estimates pair-wise transformation between two maps

Input: 2 maps represented as point clouds
Output: transformation estimate between 2 maps

1: **procedure** ESTIMATETRANSFORM($map1, map2$)
2: down-sample to working resolution
3: remove outliers
4: estimate surface normals
5: detect keypoints
6: compute descriptor for each keypoint
7: match descriptors and compute the initial transformation
8: refine transformation with ICP
9: **end procedure**

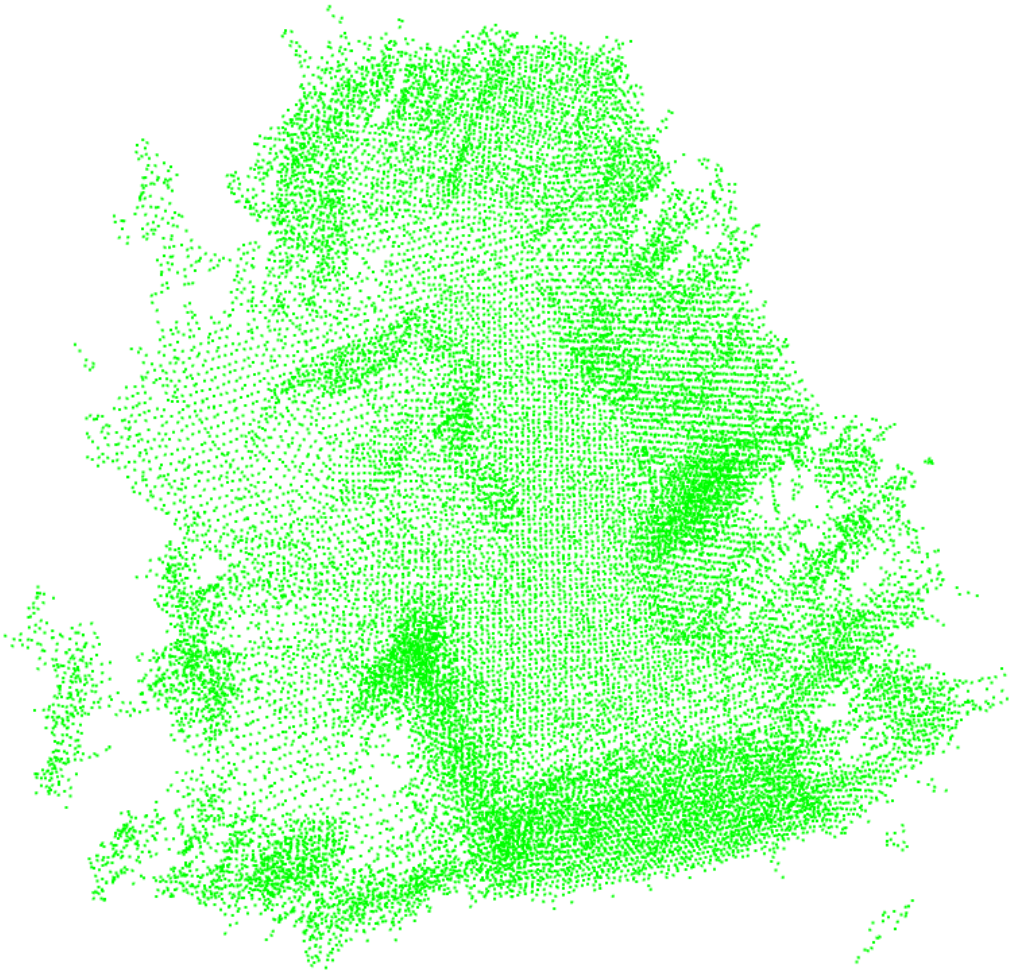


Figure 2.1: Point cloud map voxelized to resolution 0.05 meter per voxel. Notice the regular spacing between points.

Algorithm for voxelization is following. For each voxel (size of the voxel is determined by the resolution) we take all points contained in the voxel and approximate them with their centroid.

As discussed in Section 1, in a multi-robot system the down-sampling is typically performed by the SLAM algorithm running on each robot before publishing the map, thus saving bandwidth of the communication. However, in a typical situation, we might want to reduce the resolution even further for the purpose of transformation estimation to reduce the computation time (for example each robot might publish a map with the typical resolution of 0.05 meters per voxel, but for estimation we work with the resolution of just 0.1 meters per voxel).

We show in Section 4 that the presented algorithm can reliably estimate the transformation for point clouds with 0.1 meters per voxel resolution.

2.1.2 Removing outliers

Although the voxelization can deal with some of the noise and inaccuracies, during experiments it has been beneficial to perform further outliers filtering to remove far laying points. Far-laying outliers may end-up being detected as keypoints, but they are usually not matched. Reducing the number of detected keypoints speeds up the later phases of estimation.

I have selected to use a simple radius-based outlier removal. This method searches for neighbours of each point within a certain radius and removes points that have below threshold neighbours count.

Because descriptors of the outlier points are based on just a few points, which don't produce robust matching candidates, and the radius outlier removal removes only the points with few neighbours, it has not been observed to reduce the robustness of the estimation.

For example on two maps from Alpen-Adria-Universität Klagenfurt (AAU) dataset (Section 4.2), the outlier removal step removes 326 and 271 points respectively (7.29%, 6.16%), but the number of detected keypoints decreases from $66, 63$ to $51, 56$ (22.73%, 11.11%). Outlier removal didn't impact the estimation process negatively.

2.1.3 Estimating surface normals

The last preprocessing step is to estimate surface normals (Figure 2.2). Surface normals are vectors perpendicular to the surface in the neighbourhood of the point. Surface normals are used in later steps to compute descriptors and by Harris keypoint detector.

The algorithm for estimating surface normals is described by Rusu [38]. The algorithm is based on the neighbourhood search. For each point neighbourhood, we do a least-square plane fitting estimation and select the orientation of the estimated normal.

The most important parameter for normals estimation is the size of the neighbourhood which is used for the estimation. This can be configured by the user.

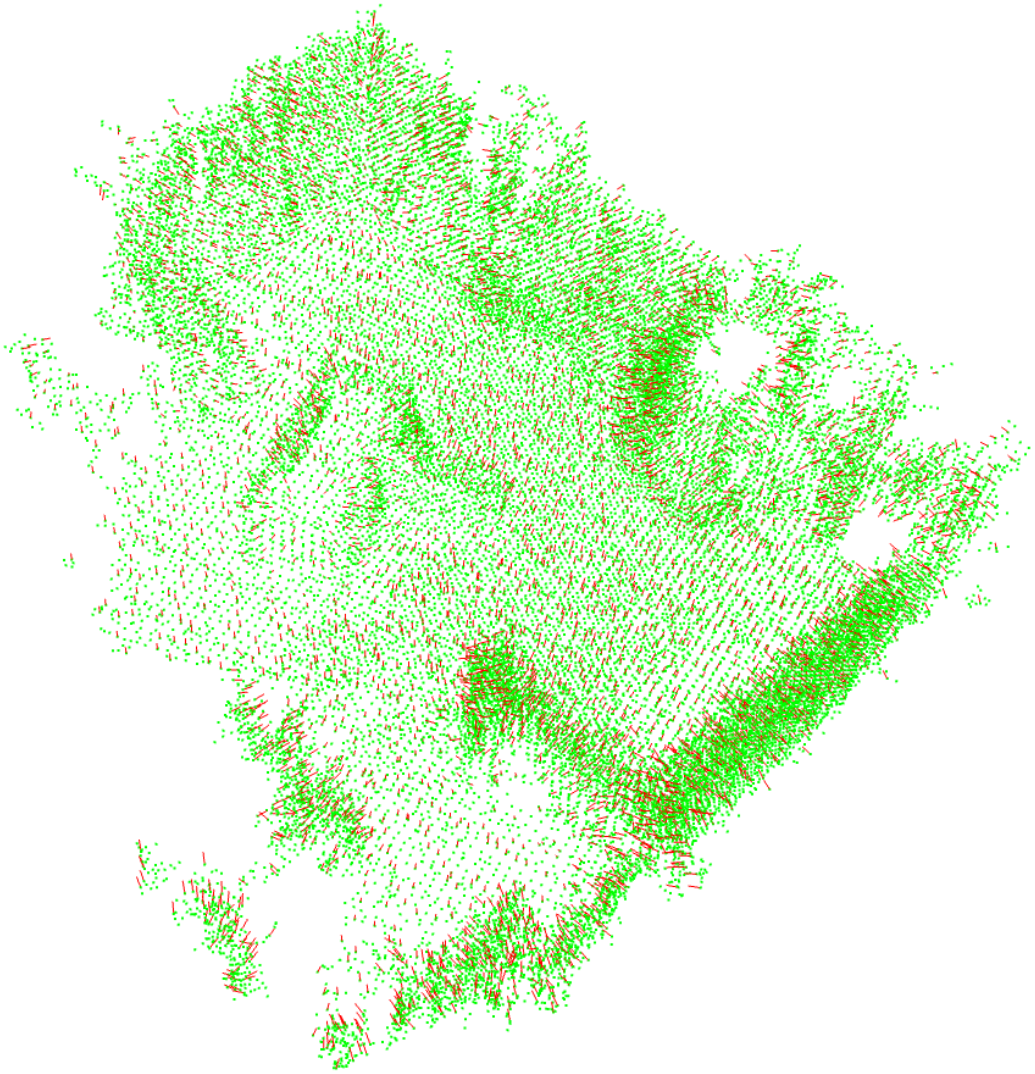


Figure 2.2: A voxelized point cloud map with estimated surface normals (red). Only each fifth normal is visualised.

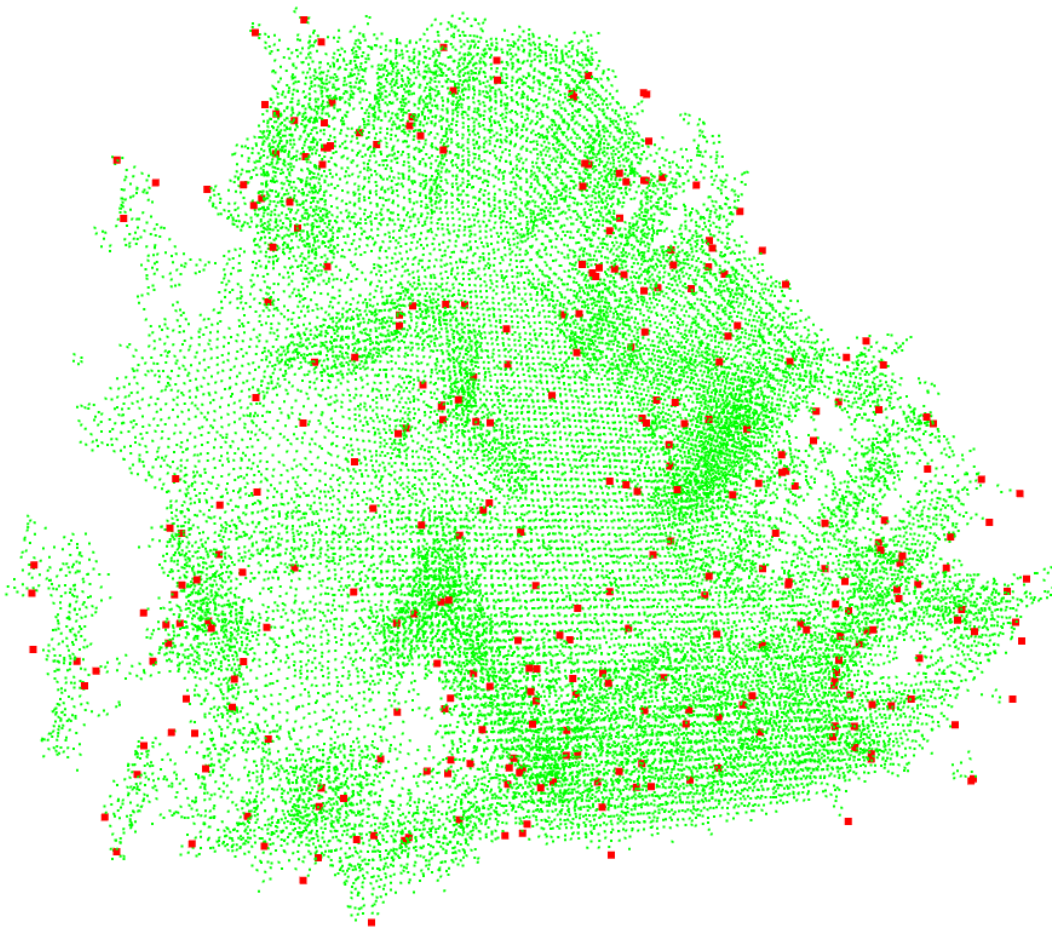


Figure 2.3: SIFT keypoints (red) detected in the point cloud map (green).

2.1.4 Detecting keypoints

Even after the down-sampling, the maps usually contain too many points to work with all of them in later steps. We need to reduce the number of points further. A common technique used in the area of computer vision and robotics is to search for keypoints that identify suitable landmark points for later matching (Figure 2.3).

There are two families of keypoints detectors that are being used with point clouds. Most of the approaches have been adapted from keypoints detectors that have been originally developed to work on images.

The first class of detectors uses RGB colour information stored for each point in the point cloud. This approach supposes that the point cloud has been obtained with a detector that provides the colour information such as a stereo camera rig, an active RGB-D camera etc. For the presented approach, I use SIFT keypoint detector, which is an adapted algorithm from Lowe [29] that works on point clouds with RGB information.

The second class of algorithms works with just geometrical information and,

therefore, is able to work with point clouds that do not store any additional information for points. These algorithms can be used with point clouds composed of laser scans. The presented algorithm implements Harris 3D keypoint detector for this purpose, which is an adapted algorithm from Harris and Stephens [19]. Instead of using image gradients, which are not available in the point cloud without colour information, it uses surface normals, that capture geometrical properties of the point neighbourhood.

2.1.5 Computing descriptors

Next step is to compute a local descriptor around each detected keypoint to be able to match keypoints between maps. Unlike the keypoint detection algorithms, algorithms for computing descriptors has been mostly designed from scratch to work on point clouds. A comprehensive review of the point cloud descriptors is given by Ali [1].

Most of the descriptors do not use the colour information and use only local geometry around the point. Widely used PFH descriptors, introduced by Rusu et al. [39], use a multi-dimensional histogram with 125 bins to provide an informative signature of a point neighbourhood. The histogram captures relationships between estimated surface normals (Section 2.1.3). It uses relationships between all pairs of points in the neighbourhood and, therefore has complexity $\mathcal{O}(k^2)$, where k is the number of points in the neighbourhood. There is a variant of PFH descriptors Point Feature Histogram with colour (PFHRGB) that stores also colour information in the extended histogram of 2×125 bins.

The main disadvantage of PFH descriptors is the high algorithmic complexity, and therefore the slow processing speed [40]. The presented implementation supports also Fast Point Feature Histogram (FPFH), introduced by Rusu et al. [40], Signature of Histograms of Orientations (SHOT) with colour, introduced by Tombari et al. [42], Radius-based Surface Descriptor (RSD), introduced by Marton et al. [30], and 3D Shape Context (SC3D), introduced by Frome et al. [14], novel descriptors which offer better processing speed compared to PFHRGB and PFH.

2.1.6 Matching descriptors

Next step in the pipeline is the descriptors matching, which yields an initial transformation estimate. It uses the features extracted from point clouds in the previous steps.

We match two sets of descriptors from two point clouds (Figure 2.4). For each descriptor from the first set, we want to find a descriptor in the second set, which describes the same place (in the second point cloud). This step is challenging because the descriptors might be less descriptive than desired. Therefore, the correct match might not be always the closest descriptor, but the k-nearest descriptor.

To overcome the issue, Sample Consensus Initial Alignment (SAC-IA) algorithm has been proposed by Rusu et al. [40] (Algorithm 2).

The algorithm combines random matching of k-nearest descriptors and the RANSAC algorithm, introduced by Fischler and Bolles [11].

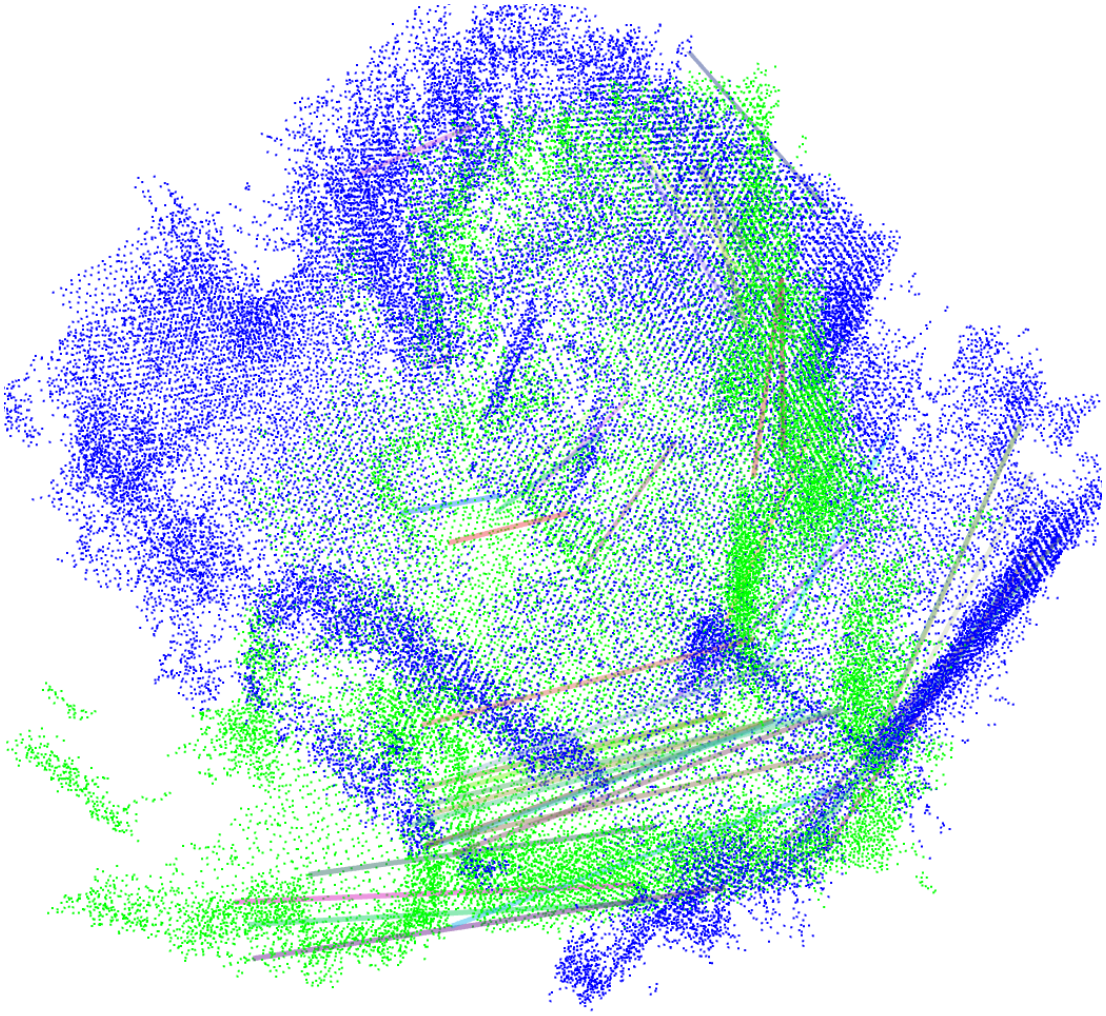


Figure 2.4: Matched keypoint descriptors between two point cloud maps (green, blue). Only inlier matches from Random Sample Consensus (RANSAC) are shown.

Algorithm 2 SAC-IA algorithm from Rusu et al. [40].

Input: D_1, D_2 set of descriptors
Output: rigid transformation estimate

function SAC-IA(D_1, D_2)
    loop repeat N times
        $K \leftarrow$ draw n descriptors from D_1
        for all $d_i \in K$ **do**
            $M \leftarrow k$-nearest matches from D_2
            $m_i \leftarrow$ a random sample from M
        end for
        estimate rigid transformation for selected samples
        determine inliers
    end loop
    return transformation with the most inliers.
end function

The algorithm was originally developed to work with FPFH descriptors, that are fast to compute, but less descriptive that PFH. With PFH and PFHRGB descriptors, SAC-IA sometimes yields sub-optimal transformation, even when there are good potential matches available in the descriptors sets because it selects matching candidates randomly instead of preferring better matches. This motivated me to develop a new matching algorithm (Algorithm 3). This algorithm is deterministic to avoid problems caused by the randomness in SAC-IA, has a good accuracy across different descriptors (Section 4.8) and requires a very little configuration (the only parameter is k), which does not need to be adjusted per descriptor.

The algorithm is based on the idea of reciprocal match validation. When we are considering match $d_i \rightarrow d_j$, we try to match also $d_j \rightarrow d_i$. We consider all k-nearest neighbours for matching to deal with potential low descriptiveness and then select the best match with reciprocal match validation. Considering k-nearest neighbours is a key idea, that can improve the robustness of the matching for point cloud descriptors compared to typically employed "1-on-1" matching with reciprocal match validation (Section 4.8). Unlike SAC-IA, the RANSAC algorithm is not incorporated in the scheme, the output of the algorithm is just matched pairs of descriptors.

Algorithm 3 My matching approach using k-nearest matches validated with reciprocal matching

Input: D_1, D_2 set of descriptors
Output: set of matches between D_1, D_2

```
1: function matchReciprocalK(D1, D2)
2:     M = {}
3:     for all d_i ∈ D1 do
4:         N ← k-nearest neighbours of d_i in D2
5:         for all d_j ∈ N do
6:             N' ← k-nearest neighbours of d_j in D1
7:             if d_i ∈ N' then
8:                 M ← M ∪ {(d_i, d_j)}
9:             end if
10:        end for
11:    end for
12:    return M
13: end function
```

Notice that our algorithm does not need any threshold for the maximum match distance and, therefore, can work with various kinds of descriptors without any need for a specific configuration. The only parameter is k – number of neighbours. As discussed in Section 4.8, my approach shows better performance than SAC-IA in most of the cases and is not influenced by the randomness.

Both SAC-IA and my reciprocal matching algorithm need to quickly find k-nearest neighbours. To achieve a good performance we use an approximative nearest neighbour search introduced by Muja and Lowe [33]. This solution allows us to match a large number of descriptors quickly, even though computational complexity rises significantly with k. During the experiments (Section 4.7), the

matching step takes usually just a fraction of time needed to extract keypoints and descriptors.

2.1.7 Estimating the final transformation

SAC-IA embeds RANSAC into its algorithm and therefore provides directly an initial transformation estimate. When we use the reciprocal matching scheme (Algorithm 3), we need to run RANSAC step with rigid transformation model after the matching to get inlier matches. In the typical scenario most of the matches are outliers, so without the RANSAC step, it is not possible to estimate a consistent transformation.

RANSAC transformation estimate is always based on the minimum number of points required to estimate the model (rigid transformation). To get a better estimate for all inlier pairs we recompute the transformation estimate using least-squares on inlier pairs. We use Singular-Value Decomposition (SVD) to compute the new estimate, the technique is described by Golub and Reinsch [16]. Another widely used technique in Computer Vision is to use a non-linear least squares approach, such as the Levenberg-Marquardt algorithm described by Moré [32], to optimize the transformation estimate re-projection error considering all inliers. Because we refine the transformation further with ICP, there hasn't been any observed difference between using Levenberg-Marquardt or SVD-based technique. The implementation uses SVD to refine the transformation on inliers.

So far we have used only detected keypoints and descriptors computed around the keypoints to estimate the transformation. To refine the transformation using all the points in the point cloud we use ICP algorithm introduced by Besl and McKay [3]. ICP algorithm tries to minimize Euclidean error distance between the estimated corresponding points (Figure 2.5). The estimated transformation from either SAC-IA or the reciprocal matching scheme presented in Section 2.1.6 is used as an initial guess for ICP.

The initial guess is usually close to the final transformation estimated by ICP, especially when using the reciprocal matching algorithm (Algorithm 3) and RANSAC for the estimation. Even though we use all the points with ICP, because of the proximity of the guess to the final transformation only a couple of iterations is needed for ICP to converge, so the refinement does not impact overall performance significantly (Section 4.7).

Since the introduction of ICP, many ICP-derived algorithms have been introduced to overcome issues associated with ICP, especially to avoid local minima. A comprehensive review is given by Pomerleau et al. [37]. Because our initial transformation is usually very close to the final transformation, the original ICP algorithm performed well for our use case during experiments.

The ICP refinement is the last step in estimating pairwise transformation. We use the output of the ICP as the final estimated transformation between two maps.

2.1.8 Evaluating the estimated transformation

To ensure robustness in real-world applications it is valuable to have a confidence measure for the estimated transformation. Commonly used measure for

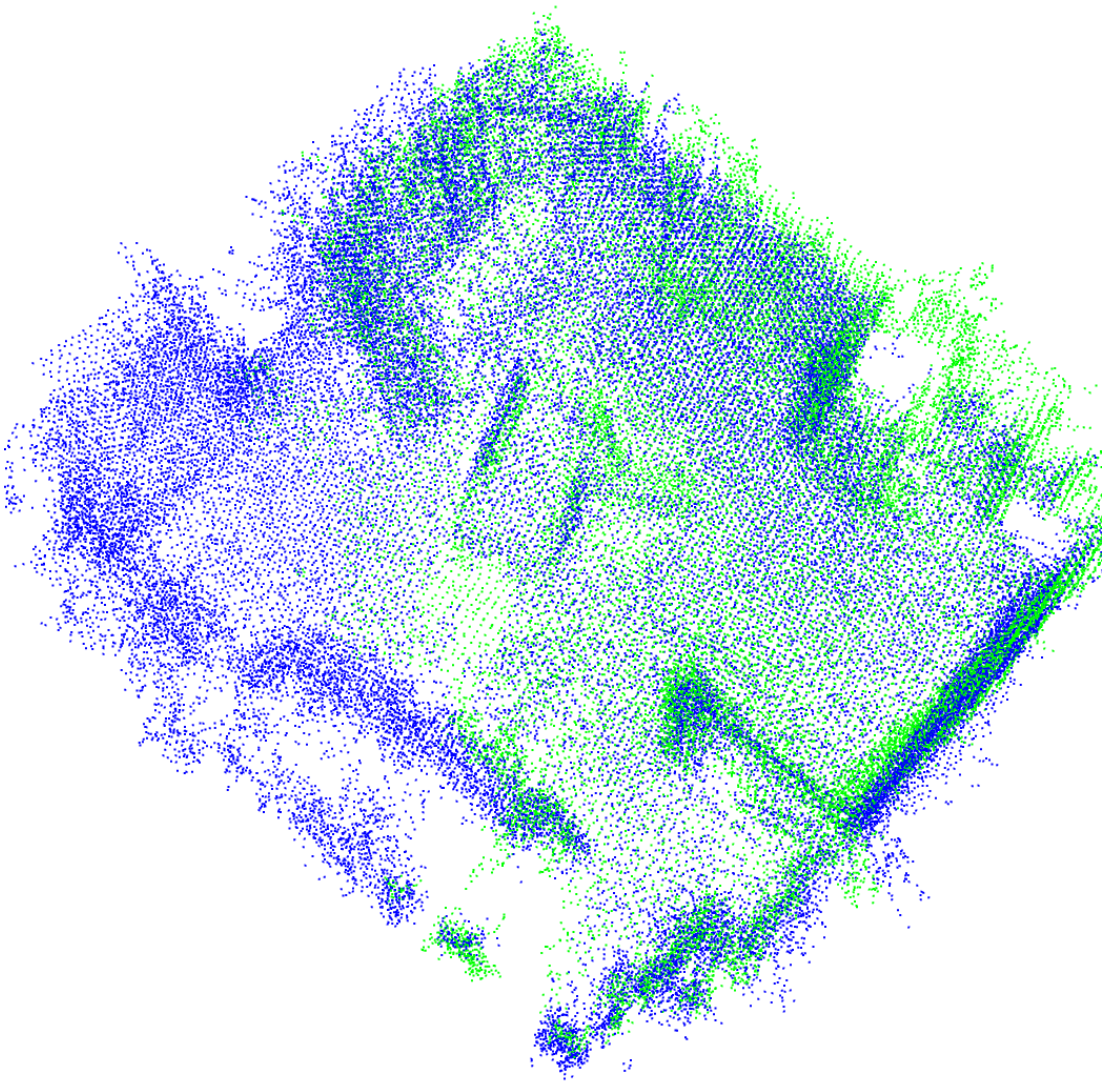


Figure 2.5: Two point clouds maps (green, blue) aligned after refining the initial transformation with ICP. The initial transformation was computed from matches shown in Figure 2.4.

RANSAC-based estimates is to use the number of inliers to access the algorithm performance. Brown and Lowe [5] proposed a probabilistic model for image match verification based on the number of inliers. The authors used a model with chosen $\alpha = 8.0$ and $\beta = 0.3$ given by Equation (2.1), where m is the number of matches and ψ is the number of inliers in RANSAC.

$$\frac{\psi}{8 + 0.3 \cdot m} \tag{2.1}$$

Another possible confidence measure is to use the error distance between the two point clouds. This approach is common with point clouds and is implemented in PCL. We typically use the Euclidean distance to have a natural metric estimate. This approach has two main issues. Because we do not know the correspondences between the two maps we need to use the nearest neighbour for each point to compute the distance. Moreover, to deal with occlusions (points outside intersection of the point clouds) we need to introduce cut-off threshold to deal with too large distances caused by points outside of the intersection.

In the implementation, I use Euclidean distance to evaluate the transformation estimate because the implementation of SAC-IA in PCL does not provide information about the inliers count.

2.2 Estimating global transformations

The map-merging problem for two maps is discussed in Section 2.1. To be able to merge more than two maps we consider a map-merging graph (Definition 2).

Definition 2 (Map-merging graph)**.** *A graph whose nodes correspond to robots' maps and whose edges represent pair-wise transformation estimates between the maps.*

To construct a map-merging graph for n maps we need to estimate $\mathcal{O}(n^2)$ pair-wise transformations. Depending on the environment and map relations the map-merging graph might be dense or sparse, but typically will be missing some edges, because some of the transformations could not be estimated, or could be estimated only with low confidence, see Section 2.1.8.

Once we have constructed the map-merging graph, the global merged map can be computed by finding a spatial configuration of the nodes (maps) that is consistent with the transformations represented by the edges. In other words, we want to find a transformation for each map from the global reference frame to the particular map consistent with the pairwise transformations.

2.2.1 Map-merging as graph-based SLAM

The idea of a map-merging graph is similar to the idea of a pose graph used in graph-based SLAM. Graph-based SLAM uses a so-called graph-based formulation of the SLAM problem. "Solving a graph-based SLAM problem involves constructing a graph whose nodes represent robot poses or landmarks and in which an edge between two nodes encodes a sensor measurement that constrains the connected poses." [18].

In the graph-based SLAM graph, we have poses at different points in time, in the map-merging graph we have maps with origins in unknown positions. In the map-merging graph, we associate our pairwise transformation estimates with edges and in the SLAM graph, we use usually direct sensor measurements. In both graphs, we want to find the spatial configuration of the nodes of the graph that is the most consistent with constraints represented by the edges. If we are able to construct the map-merging graph (for which we need to do $\mathcal{O}(n^2)$ pair-wise transformation estimates), we can apply the graph-based SLAM techniques on the map-merging graph to find the spatial configuration of the nodes that is consistent with our pair-wise estimates. Typically, the Gauss-Newton algorithm, described by Fletcher [12, ch. 3], or the Levenberg-Marquardt algorithm, described by Moré [32], are used to optimize the pose graph in graph-based SLAM.

To support graph-based SLAM approaches, there are libraries specifically focusing on graph optimization under constraints. Well-established examples include `G2O`, developed by Kümmerle et al. [26], and `TORO`, developed by Grisetti et al. [17]. These libraries can be leveraged to solve the map-merging problem for n maps.

2.2.2 Solving map-merging problem without loop closures

Graph optimizing techniques benefit from the presence of loop closures in the graph. During SLAM mapping a loop closure occurs when the robot revisits a node in the graph. Loops in the map-merging graph require the robot to be able to estimate pair-wise transformation with at least two neighbours. For example for the shortest loop in the graph, a triangle with three robots, we need each robot to be able to estimate both pair-wise transformations to remaining robots.

Loops in the map-merging graph will usually only occur in fairly large environments and when using many robots. Further on, unlike in graph-based SLAM, loop closures may not be critical for good performance of the map-merging algorithm in the typical setup. In the SLAM graph, we expect measurements associated with the edges to be subject to the Gaussian noise and corrections based on loop closures are usually vital for good SLAM performance. In the map-merging graph, we have pair-wise estimates associated with edges based on the geometry of whole maps containing a large number of points. The pairwise estimates tend to be quite precise because they consider large portions of the environment. There are also available evaluation metrics for the pair-wise estimates (see Section 2.1.8), which allow us to use only robust estimates in the graph.

Taking advantage of accurate pair-wise estimates in the map-merging graph we may avoid estimating all $\mathcal{O}(n^2)$ pair-wise estimates, especially since the pair-wise estimation is the most time-demanding step. To save time we can avoid pair-wise estimates introducing loops in the graph. We can stop pair-wise estimation when the nodes are within one connected component of the graph. We may want to employ a suitable heuristic for selecting the order of pair-wise estimates. Using this approach we might need only $\mathcal{O}(n)$ pair-wise estimates to estimate global transformations for the maps.

Below I present fast, non-iterative algorithm (Algorithm 4) for extracting the global poses of maps from the map-merging graph suitable for use in such setup. The algorithm does not take any advantage of the loops in the graph (we

expect the use of the approach mentioned above, which avoids loop closures). The algorithm, however, can work with graphs containing loops.

Algorithm 4 The algorithm to extract global poses of the maps from the map-merging graph (Definition 2).

Input: weighted map-merging graph
Output: Transformation for each map from the global reference frame to the map reference frame

function EXTRACTGLOBALPOSES($G = (V, E)$ map-merging graph, $P_e \forall e \in E$ pair-wise transformation estimates)
  $C = (V_C, E_C) \leftarrow$ the largest connected component in G
  $T \leftarrow$ maximum spanning tree of C using confidences as weights
  $r \leftarrow$ any node from T, selected as the reference frame
  $E_s \leftarrow$ edges of T sorted by the distance from r
  $A_r \leftarrow I$
  for all $(i, j) \in E_s$ **do**
    $A_j \leftarrow A_i P_{(i,j)}$
  end for
  $\forall i \notin V_C : A_i \leftarrow 0$
  return $A_0, A_1, \ldots A_n$
end function

Because only the maps in the same component of the map-merging graph can be related to the same reference frame, the first step in the algorithm is to extract the largest connected component from the graph. In our setup we expect robots to ultimately form one component and the transformations are computed only for this component. Remaining transformations are set to null transformation. If the robots are expected to operate separated in multiple components for a long period, it might be beneficial for the coordination to compute transformations in all the components, each component having its own reference frame.

Next step in the algorithm is to extract the maximum spanning tree. The edges are weighted with estimated confidence for each pair-wise transformation so that the most confident estimates are used. The possible loops are removed in this step and the non-tree edges are not used.

We can select any node from the spanning tree as the future global reference frame. All the transformations will be computed to this reference frame.

The last step is to compute the transformations to the selected reference frame. For computing the transformation of the node (map) we need transformations on the path from the reference frame (tree root) to the current node to be computed first (actually we need just the preceding transformation, but that implies computing all the transformations). To achieve this we can sort the edges by the distance from the reference node, or just traverse the tree using Breadth-First Search (BFS) or Depth-First Search (DFS) and compute the transformations during the traversal.

3. Implementation

I have implemented the map-merging algorithm (Chapter 2) in the ROS framework (Section 3.1). This allows the implementation to be used with various readily available SLAM algorithms and inter-robot communication solutions.

Point cloud algorithms are implemented with PCL (Section 3.2). PCL is a popular open-source library for manipulating point-cloud data.

3.1 Robot Operating System

ROS is a popular open-source robotics middleware and distribution of community-maintained robotics software. ROS provides hardware abstraction, low-level sensors control, implementations of commonly-used robotics algorithms, including several variants of SLAM, message-passing between processes, build system, and package management.

The core of the ROS is a messaging system, that allows loose coupling between sensors, robotic algorithms and actuators. There are many community-established message formats for various sensor types, visualisation, maps, robot state, linear algebra etc., which enable software reuse.

Software in the ROS project is organised in a form of community-maintained packages. The first type of packages integrate particular hardware sensors, actuators or entire robot platforms with ROS and provide a message interface using standard ROS messages. The second type of packages implements robotics algorithms and usually depends only on standard ROS messages, making them reusable between robots. The ROS project builds and distributes binary packages for Linux.

3.2 Point Cloud Library

PCL is an open-source library for point cloud processing. It contains routines for feature estimation, surface reconstruction, registration, the nearest neighbour search, model fitting, including RANSAC algorithm, and segmentation. The library offers serialisation of point clouds to its native, but widely supported `pcd` file format. PCL is supported in ROS, with PCL point cloud format being supported for message serialisation into ROS messages.

I have used PCL to implement the presented map-merging algorithm (Chapter 2) and to read and save `pcd` files for the command-line tools (Section 3.3.5).

3.3 `map_merge_3d` package

The software for this thesis is organised in a ROS package `map_merge_3d`. This package contains ROS node for online map-merging as well as command-line applications for offline map-merging and visualisation of point cloud files.

The `map_merge_3d` package is distributed within ROS starting from ROS Melodic release. The package documentation is maintained as wiki text available online, a reproduction of the documentation is attached (Appendix A).

Care has been taken to integrate the package with the rest of the ROS ecosystem. The package is not tied to any particular package for inter-robot communication (Section 3.3.1) and it depends on the commonly used map representation in ROS (Section 3.3.2).

The ROS node supports auto-discovery of the robots (Section 3.3.3) and is designed to partially mitigate the high computing demands of the map-merging (Section 3.3.4). The package also contains command-line tools for offline map-merging and visualisation (Section 3.3.5).

3.3.1 Communication

The ROS node is designed to work with any software for inter-robot communication. This architecture allows users to use any kind of communication media and software solution that is available for a given multi-robot system. The package, therefore, does not provide nor expect any particular communication between robots.

To achieve the loose coupling with communication software, the node expects map topics to be available within the local ROS graph. A user of the package is then expected to configure communication between robots that publishes robot maps within the local ROS graph. There are several solutions available in ROS to achieve this setup.

First of all, ROS natively supports sending messages between multiple computers connected to the same network. However, the ROS master directory listing (broker) service is not distributed and it needs to run on one of those computers. This creates a single-point-of-failure that is not acceptable for unreliable communication media.

To achieve reliable multi-robot communication ROS Multimaster Special Interest Group was formed with the intent to support running multiple ROS master broker services. Juan and Cotarelo [25] developed the `multimaster_fkie` package, that deals with most of the problems of the multi-robot communication. This package can be used together with `map_merge_3d` to run online map-merging for multiple robots.

Users are free to choose any other communication solution meeting their needs for their particular application, `map_merge_3d` does not depend on any particular multi-master setup and can run also under single-master setup, for example in a simulated multi-robot environment.

3.3.2 Map representation

Choosing a correct map representation is important to allow interoperability and re-usability within the ROS ecosystem. We need to select a map format that is supported by the most SLAM implementations in ROS to be able to consume maps from the robots.

Widespread 3D SLAM approaches in ROS produce maps as `sensor_msgs/PointCloud2` messages, which are now de-facto standard for representing 3D maps in ROS. This representation is suitable for the presented map-merging algorithm (Chapter 2).

Unfortunately, there is no standard message format in ROS to represent maps as a pose graph of point clouds, which would allow implementing map-merging on point cloud pose graph discussed in Section 1.4. The point cloud pose graph format allows repairing mapping errors.

The future standardised map format may take inspiration from Google's Cartographer SLAM introduced by Hess et al. [20]. This 2D SLAM works on a graph of occupancy grid *submaps*. This format would be suitable for the error correcting approach (Section 1.4). The format would need to be adapted to support 3D maps or it could be extended to support both 2D and 3D maps. Standardising the message format in ROS would require a community discussion to ensure that the format can be supported across different SLAM approaches. While some SLAM implementations in ROS export their internal SLAM graph, to the best my knowledge, Google's Cartographer is the only implementation in ROS that offers a suitable pose graph map format that would be suitable for the map-merging on pose graphs. However, the offered map format is custom to Cartographer SLAM and it is supporting only 2D maps as Cartographer is a 2D SLAM, so it cannot be used in this work.

The map-merging node, therefore, uses monolithic maps represented in `sensor_msgs/PointCloud2` messages, which always encode the whole map, to allow the highest degree of interoperability with existing ROS packages.

Apart from not being able to correct mapping errors, a monolithic map representation also causes unnecessary transfers of already-explored space, that slows the map update rate for large-scale maps. A standardised pose graph map representation in ROS might help to solve both issues.

3.3.3 Configuration

The ROS node supports merging maps from an arbitrary number of robots. To make configuration of the node easy, maps for map-merging are auto-discovered by the ROS node.

The node periodically scans the ROS graph to discover map topics, which are then subscribed and passed to the map-merging procedure. This mechanism also enables robots to be added to the multi-robot system later, for example, to tackle a high demand for the service.

Full documentation of this mechanism is described in Appendix A.

3.3.4 ROS node architecture

Estimating transformations between maps with the presented map-merging algorithm (Chapter 2) is computationally intensive and can take a long time, especially when merging many large-scale maps.

To achieve high update rates of the merged map, the ROS node uses an asynchronous architecture. Because the map origins do not change, we don't need to re-estimate transformations between maps every time we are updating the merged map, we can use the previous estimates to update the map.

The node is designed to run periodically asynchronous map composition and transformation estimation tasks, which are independent, at user-settable frequencies. The map composition uses already estimated transformations between maps,

transforms the input maps and concatenates the maps to produce the merged map. This process is fast and allows the node to achieve high update rates of the merged map to quickly incorporate newly-discovered areas by the robots.

The transformation estimation task uses a computationally demanding algorithm (Chapter 2) to find the transformations between maps with enough overlapping space. This task can run with a lower frequency to save computational resources.

3.3.5 Offline map-merging and visualisation

The ROS package contains also two command-line applications besides the main ROS node. The applications work with point clouds saved in `pcd` files, a point cloud file format of the PCL library.

The first application, `map_merge_tool`, is designed for the offline map-merging. It accepts n point cloud files and produces a single merged map saved to a file. It uses the same algorithm for map-merging as the ROS node. This application may be used to create a complete global map of a large-scale environment for deployment of a robotic system after a multi-session mapping of the environment with a single robot.

The second application, `registration_visualisation`, accepts two point clouds and visualises pair-wise transformation estimation algorithm as described in Chapter 2. The tool is mostly controlled by the command-line arguments, but the visualisation window is graphical. Each step of the estimation is visualised in 3D, the user is able to freely navigate the point clouds or save the visualisation as an image. This application is intended for estimation parameter tuning and learning about the map-merging process.

4. Evaluation

The presented map-merging algorithm (Chapter 2) has been evaluated on a number of demanding robotics datasets. The datasets include data captured by small aerial vehicles (Sections 4.1, 4.2) as well as ground-based robots. The datasets include both widely used benchmark datasets in robotics research and data recorded by the author.

Sensors used include stereo rig cameras and active RGB-D cameras, which are popular visual sensors in the mobile robotics. This variety of sensors and robots covers many typical multi-robot deployments. All datasets have been captured under real-world conditions, none of them uses simulated data.

The evaluation focuses on properties of the presented pair-wise transformation estimation algorithm for point clouds (Section 2.1), which is the core algorithm of the implemented map-merging ROS node (Section 3.3). The accuracy of the estimation algorithm is critical for the overall map-merging process.

4.1 The EuRoC micro aerial vehicle dataset collection

The publicly available datasets introduced by Burri et al. [6] was collected onboard a micro aerial vehicle (Figure 4.1) equipped with a stereo camera rig and an Inertial Measurement Unit (IMU). Calibration data for the cameras and ground-truth data are provided with the dataset. This dataset has been used extensively by researches for evaluation of the visual SLAM algorithms and visual odometry approaches.

4.1.1 Dataset description

The cameras produce a WVGA monochrome (greyscale) images at 20 frames per second. Cameras have a global shutter. The automatic exposure control is independent for both cameras. According to the published errata [6], this "resulted in different shutter times and in turn in different image brightnesses, rendering stereo matching and feature tracking more challenging".

The dataset contains 11 mapping sessions in 3 different environments ("Machine Hall", "Vicon Room 1", "Vicon Room 2"). Each mapping session is available in a single ROS bag file. First 5 sessions were captured in ETH machine hall (Figure 4.2), a fairly large industrial environment featuring piping, reservoirs and different types of surfaces. The second and the third batch of datasets were captured in a smaller furnished rectangular room. For the second and the third batch, the furnishing was different.

4.1.2 Maps generation

The dataset is intended for evaluating SLAM algorithms, for our purposes, it was necessary to process the data with a SLAM algorithm to create a point cloud maps.

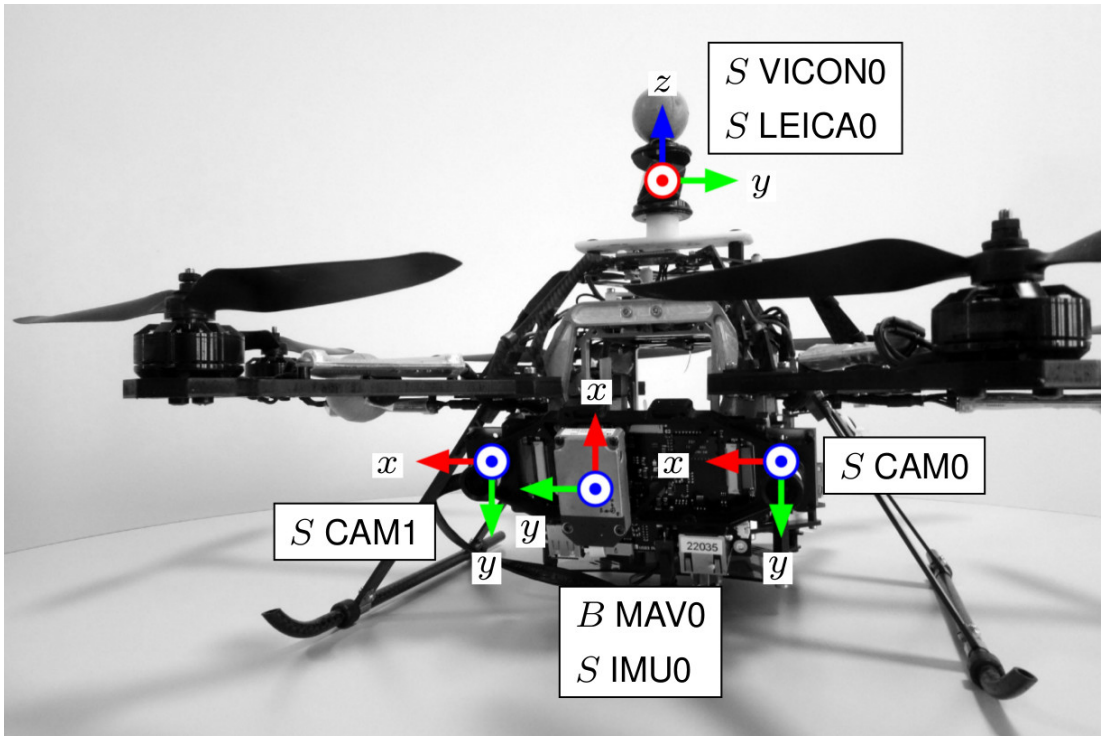


Figure 4.1: An Asctec Firefly hex-rotor micro aerial vehicle used for collecting the EuRoC micro aerial vehicle datasets. The picture shows reference frames of the sensors – the stereo cameras and IMU. The image is courtesy of Burri et al. [6].

Figure 4.2: ETH Machine hall industrial environment where 5 mapping sessions of EuRoC dataset were captured. The image is courtesy of Burri et al. [6].

First, I have used the provided calibration sequence and I have created a calibration data for ROS using the `camera_calibration` tool available with ROS.

Second, for each environment, I created a pair of maps using the "01" and "02" mapping sessions from the datasets, which were used further for the evaluation of the map-merging (Figures 4.3, 1.1, 4.4).

Based on evaluations of SLAM methods in ROS done by da Silva et al. [9] and Ibragimov and Afanasyev [24], I used RTAB-Map SLAM, developed by Labbé and Michaud [27], to create the maps. RTAB-Map can work with both stereo cameras and RGB-D sensors and produces dense maps of good quality, as noted by both da Silva et al. [9], Ibragimov and Afanasyev [24]. "RTAB-Map computed coherent SLAM solutions on all evaluated datasets and thus can be considered an efficient solution for 3D mapping scenarios", summarise the results of RTAB-Map da Silva et al. [9].

The odometry for mapping was provided from stereo camera data, using a visual odometry approach, the available IMU data were not used. For the "01" sessions I have used a native visual odometry approach of RTAB-Map. For the "02" sessions I have used visual odometry approach of ORB-SLAM2, introduced by Mur-Artal and Tardós [34] because the data exhibit more dynamic motions, which lead to a frequent loss on visual odometry using the first approach. ORB-SLAM2 visual odometry exhibited better robustness for this data. In both cases, a loop-closure approach of Labbé and Michaud [27] has been used.

The maps have been voxelized to the resolution of 0.05 meters per voxel, which yields suitable map sizes for map exchange in a multi-robot system.

Using a different SLAM pipeline for each of the maps contributed to introducing different mapping errors and artefacts, which makes map-merging of such maps more difficult.

4.2 AAU dataset

The dataset recorded at Alpen-Adria-Universität Klagenfurt (AAU) on-board a micro aerial vehicle in an outdoor forest environment. The vehicle was equipped with a stereo camera rig and IMU. The cameras produce greyscale images.

This dataset contains two mapping sessions. The environment consists or medium-sized trees, the ground is covered with leaves. The lighting conditions are challenging as there are areas of direct sunlight as well as areas in the shade. The lighting conditions are similar in both sessions, as the sessions were captured in the similar time of day. This setup causes difficult conditions for the stereo matching and the pose estimation, introducing mapping errors into the maps, which in turn make the map-merging a challenging task.

Maps (Figures 4.5, 4.6) were generated in the similar manner as described in Section 4.1.2. Mapping has been done with RTAB-Map SLAM [27] using its native visual odometry approach.

4.3 MFF dataset

This dataset consists of my own experiments conducted at the campus of the Faculty of Mathematics and Physics, Charles University. I have conducted two

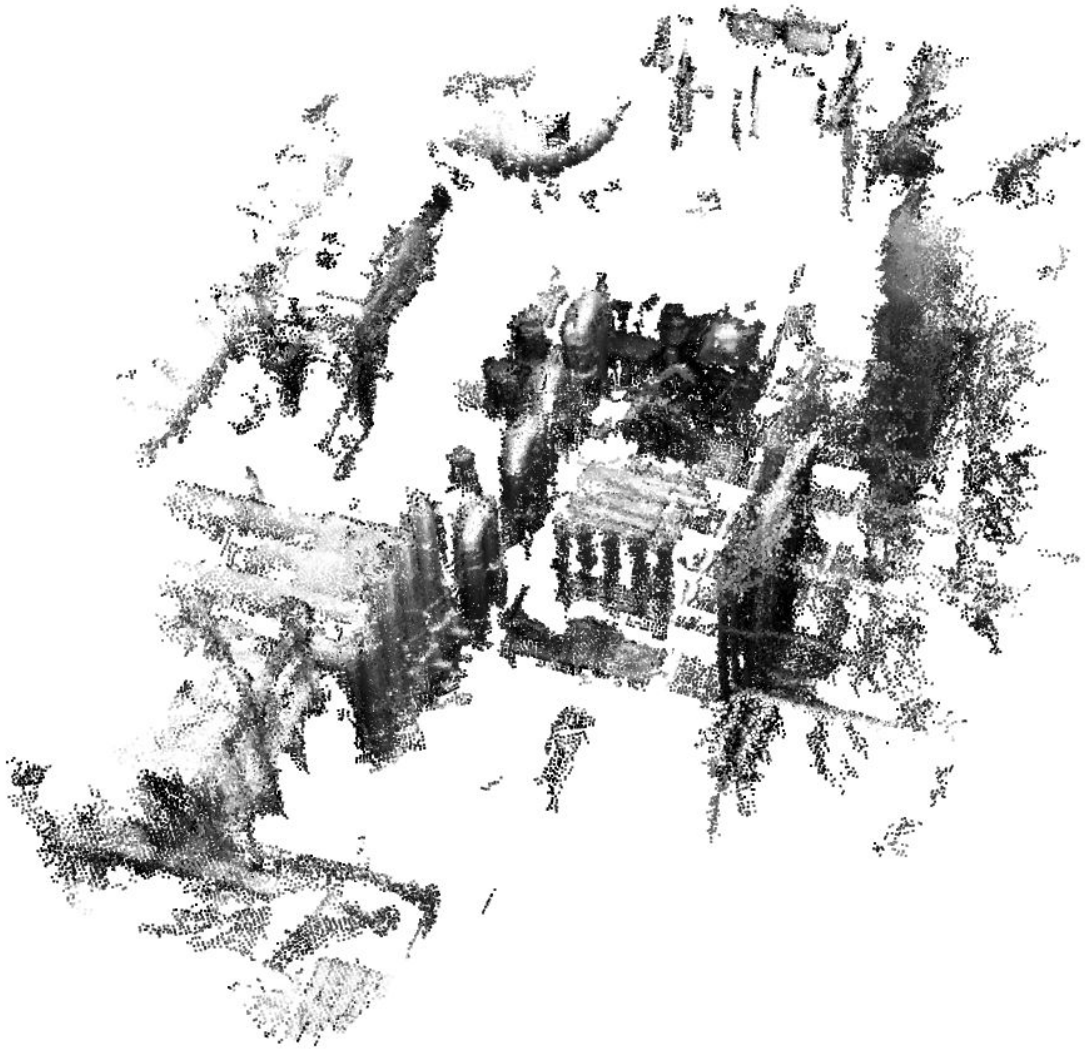


Figure 4.3: A point cloud map of the ETH Machine hall. The map was produced from "Machine Hall 02" dataset using an ORB-SLAM2 visual odometry [34] and RTAB-Map loop-closure approach [27].

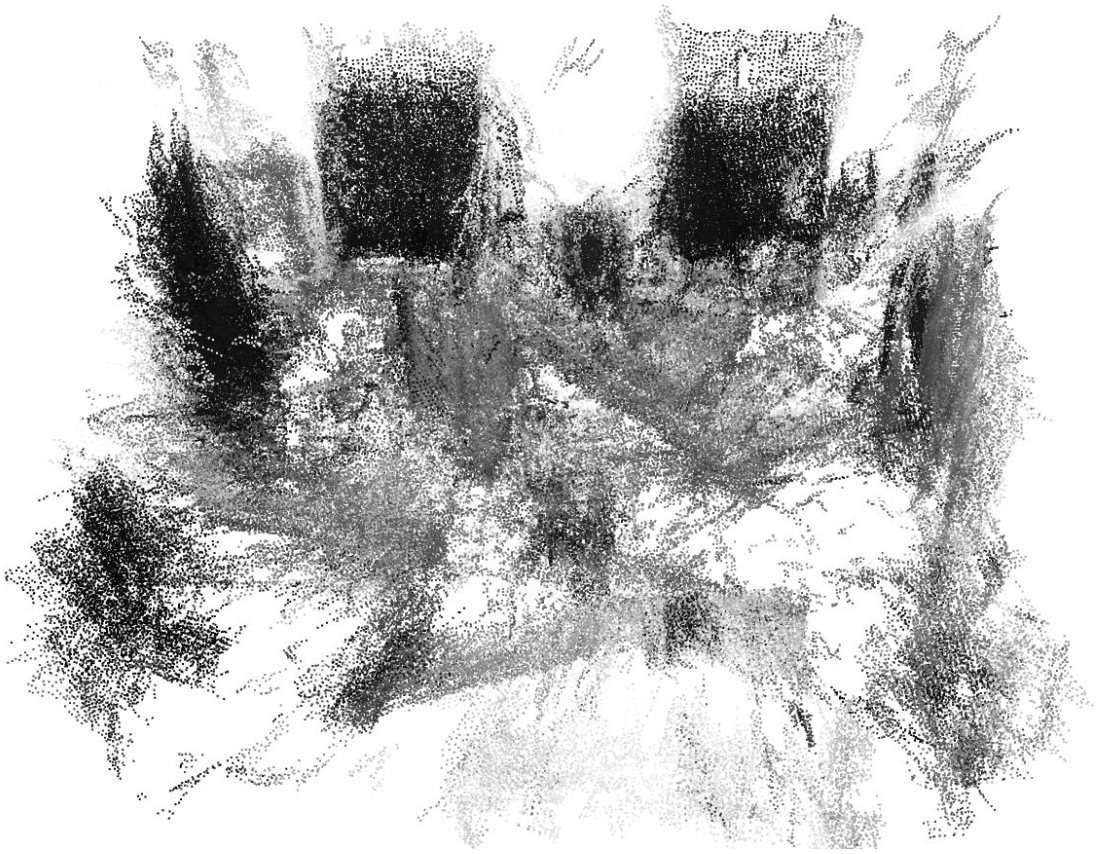
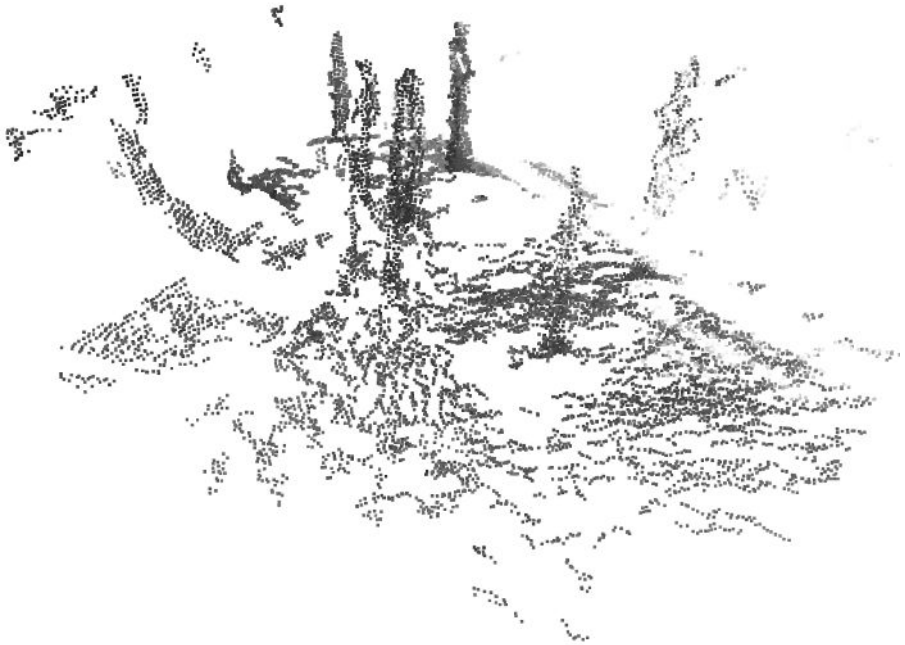


Figure 4.4: A point cloud map of “Vicon Room 2”. The map was produced from “Vicon Room 2 02” dataset using an ORB-SLAM2 visual odometry [34] and RTAB-Map loop-closure approach [27].



(a) “AAU forest 1” map

(b) “AAU forest 2” map

Figure 4.5: Top view of the two point cloud maps from AAU outdoor forest dataset. Notice the stripe of direct sunlight on the ground causing difficult conditions for stereo matching.

(a) “AAU forest 1” map

(b) “AAU forest 2” map

Figure 4.6: Lateral view of the two point cloud maps from AAU outdoor forest dataset. In both maps, there is a non-flat terrain and number of outliers caused by branches and difficult mapping conditions.

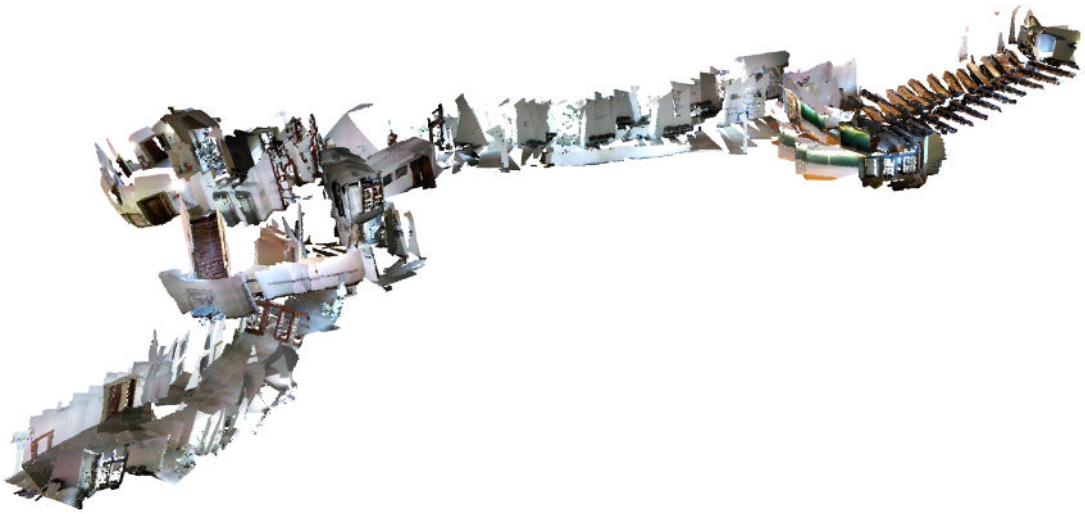


Figure 4.7: The point cloud map of the second and third floor of the faculty building recorded during handheld mapping. The visual odometry was reset immediately after the tracking was lost. Due to frequent visual odometry failures, the mapping attempt was not successful, although there are some partially consistent areas within the map.

series of experiments.

The first series of experiments was using a handheld Orbbec Astra RGB-D camera. I intended to create multi-floor maps of the campus building and evaluate this scenario with the map-merging algorithm. It has soon become apparent that without odometry source such mapping is challenging even for state of the art visual odometry approaches (Figure 4.7).

Typically, during the mapping in narrow spaces, in difficult lighting conditions or during fast movements, the visual odometry tracking is lost and needs to be reset. Without the IMU or any other external odometry source to bridge the loss of the visual odometry, it is hard to maintain a plausible odometry estimate. Two situations have been observed to be especially challenging for visual odometry approaches: rough movements (especially rotations) typical for handheld mapping, as humans tend to do less disciplined movements than robots, and transitions between spaces through narrow passages (e.g. doors), especially between spaces with different lighting conditions. Orbbec Astra camera does not contain an IMU and, therefore, is less suitable for the handheld mapping, which typically exhibits rough movements.

Bridging the visual odometry gaps with SLAM loop-closures is not always possible, especially for the second aforementioned type of errors. If the odometry is always lost when moving in and out of the specific room, such room became an isolated part of the SLAM graph without any connections to the rest of the graph. Loop-closures between these two spaces connected by the narrow passage are very hard to achieve.

Because of the presented issues, I have conducted the second series of experiments with a TurtleBot2 robot. It is a ground-based robot research platform designed to operate on flat surfaces.

The robot was equipped with Asus Xtion PRO RGB-D camera, which was used for mapping, a gyroscope and wheel encoders providing 2D odometry.

The datasets used for evaluation were recorded in two different spaces "Rotunda", a half-circle computer laboratory with adjacent rooms (Figure 4.8), and "Refectory", containing a baroque refectory and lecture rooms at the first floor of the faculty building (Figure 4.9). In each of the environments, two maps were recorded.

The maps were generated by RTAB-Map SLAM as in the previous datasets. I used Extended Kalman Filter (EKF) to fuse odometry data from the gyroscope, wheel encoders and the visual odometry provided by the native RTAB-Map approach. I used an EKF implementation for ROS by Moore and Stouch [31]. The filtering was using a 2D assumption as the robot operates in the plane. Together, these data provided a reliable odometry source for the SLAM algorithm. The resulting maps were voxelized to the resolution of 0.05 meters per voxel. The maps are examined in Section 4.9.

4.4 Accuracy

For all the aerial datasets, the map-merging is able to correctly merge the maps. The Euclidean error scores, computed as per Section 2.1.8, are small across the datasets (Figure 4.10). There are larger errors for the "AAU forest" and "Machine Hall" datasets caused by the outliers and mapping errors in those datasets, as those datasets have been recorded in tough conditions. Considering the artefacts in the datasets, the alignment is almost perfect and would allow precise coordination of the robots.

Both "Vicon Room" datasets show very small Euclidean error, well beyond the size of one voxel (0.05 m). Considering the maps are down-sampled for the registration to the resolution of 0.1 meters per voxel, the presented algorithm can achieve a sub-voxel accuracy. The registration resolution of 0.1 meters per voxel, which enables merging of larger maps in real-world conditions, was used for all the evaluated maps.

The Euclidean error for the initial estimate is significantly lower for the presented reciprocal matching method (Section 2.1.6). For the SAC-IA algorithm, the error is higher because the randomness used in the algorithm introduces additional errors in case of well-performing PFH descriptors. The initial estimate produced by the presented matching method is close to the final estimate refined with ICP. The ICP refinement is then very fast without negative impact on the overall time of the map-merging algorithm (Figure 4.14).

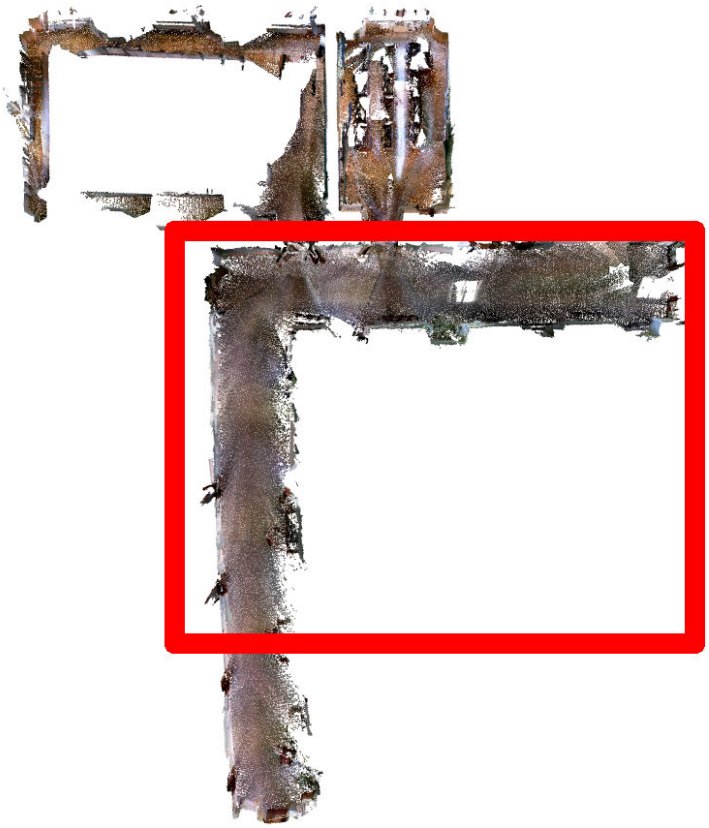
4.5 Estimation robustness

An important measure of the estimation robustness is the number of inliers (Figure 4.11) in the RANSAC estimate (Section 2.1.8). Inliers are the points which are ultimately used for computing the initial transformation estimate. Estimates based on a few inliers might be severely affected by the noise. On the other hand, estimates based on a large number of inliers, especially with the high inlier ratio (the number of inliers to the number of matches), are supposed to be reasonably confident in typical applications.

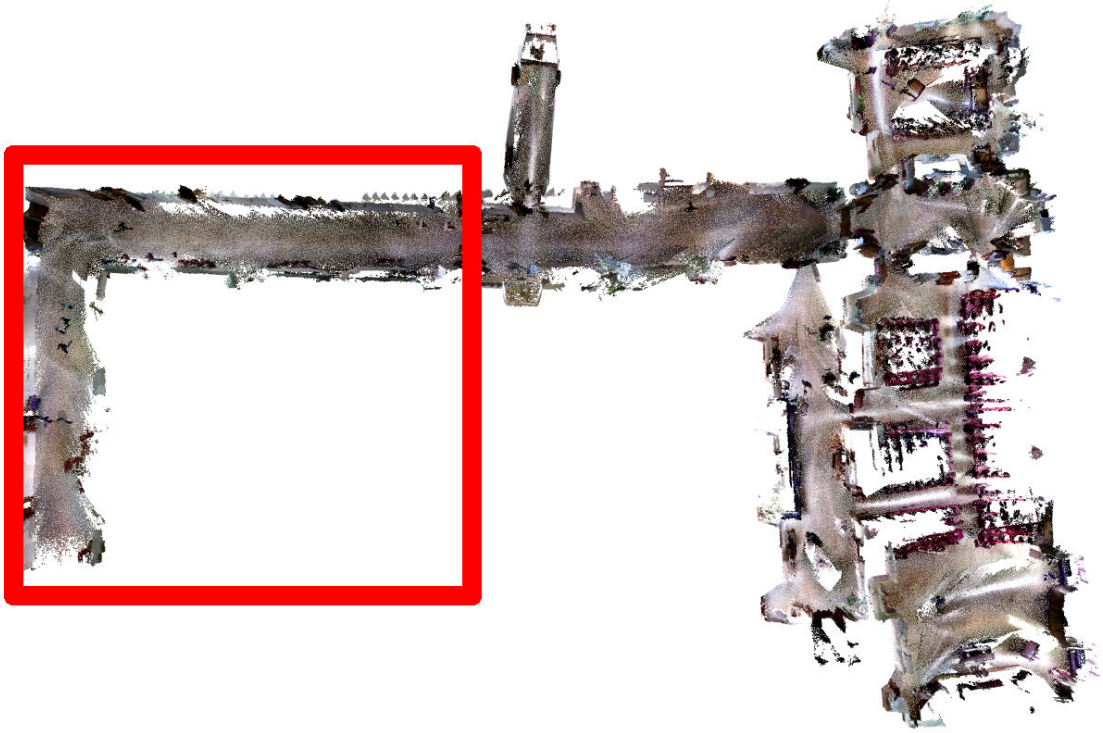
(a) “MFF Rotunda 1” map

(b) “MFF Rotunda 2” map

Figure 4.8: Maps of “Rotunda” computer laboratory with adjacent rooms recorded at the faculty building. The maps share a common mid-size area of a symmetrical half-circular laboratory. Both maps contain several mapping errors.



(a) “MFF Refectory 1” map



(b) “MFF Refectory 2” map

Figure 4.9: Point cloud maps recorded on the first floor of the faculty building. The common area is highlighted. The larger map contains a baroque refectory with adjacent areas while the smaller map contains two lecture rooms. Notice the mapping errors in the corridor area, especially in the larger map.

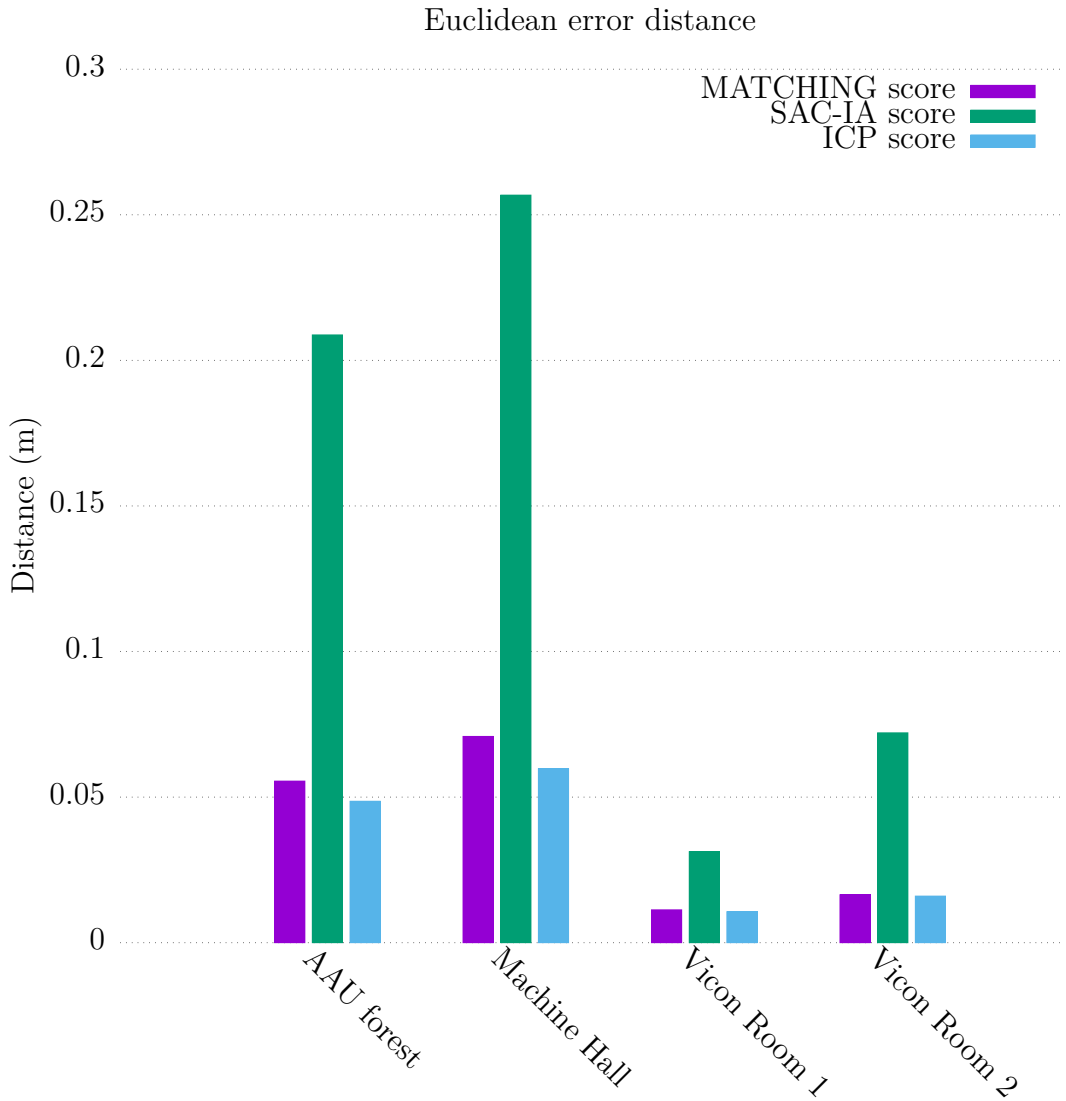


Figure 4.10: Euclidean error scores (the sum of Euclidean distances for corresponding points) across the datasets. A threshold of 1.0 has been used to cut-off fair laying point in non-overlapping areas, see Section 2.1.8 for details. The map-merging run with default parameters including PFH descriptors and SIFT keypoints. The first two scores are for the initial estimation, using my reciprocal matching scheme (Algorithm 3) and the SAC-IA algorithm respectively (Section 2.1.6). The last score is the Euclidean error after final refining with ICP (Section 2.1.7). The ICP refinement uses the initial estimate from the reciprocal matching.

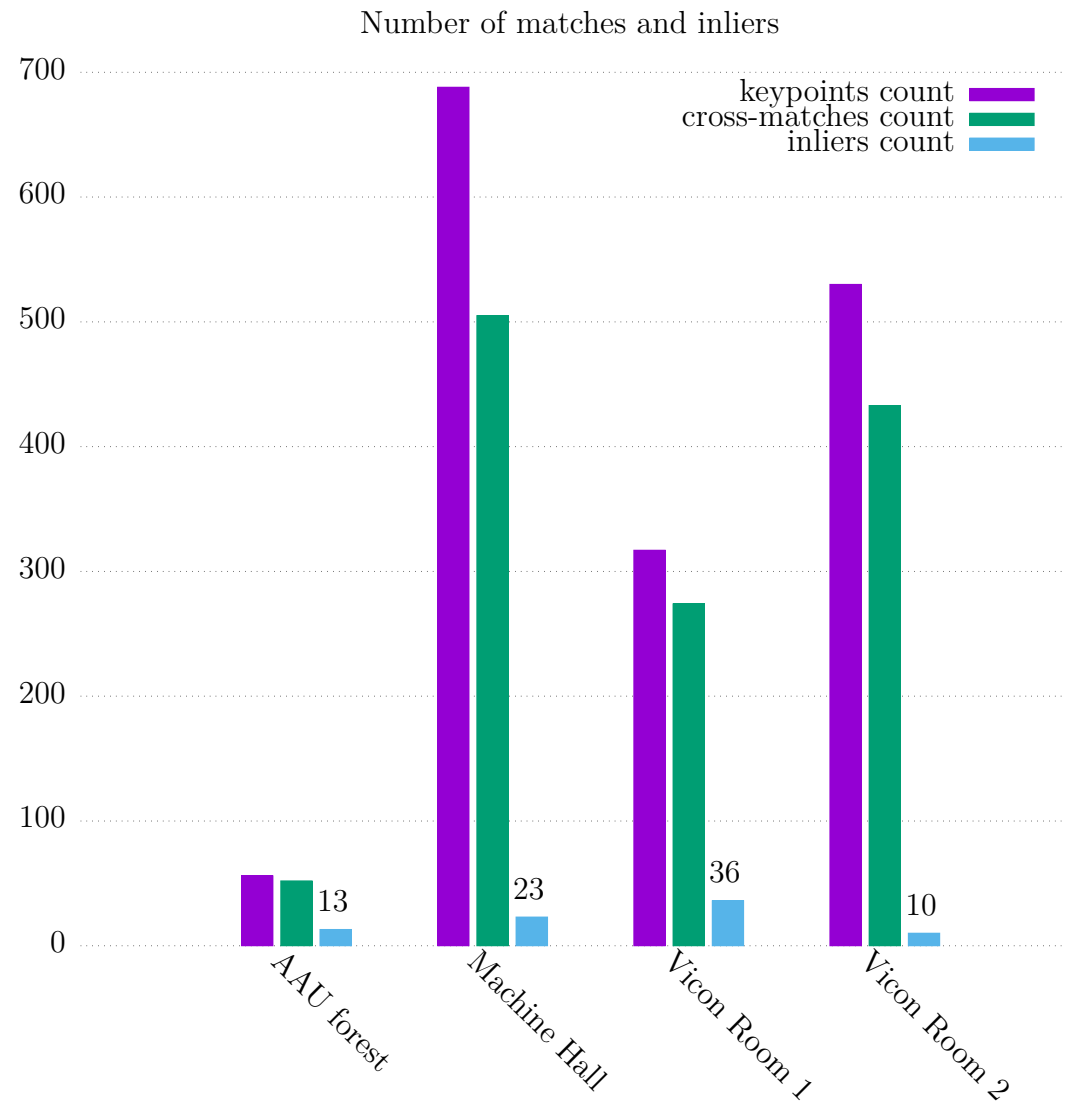


Figure 4.11: The number of keypoints, descriptor matches and the number of inliers across the datasets with default parameters. Default parameters include PFH descriptors and SIFT keypoints. The reciprocal matching algorithm (Algorithm 3) has been used for matching. Keypoints are shown for the first map of each environment.

The inlier ratio is quite low, even for PFH descriptors (Figure 4.11), the similar ratios have been observed for all the descriptors. This is partially caused by the matching scheme designed to accept also more distant matches to improve inlier ratio for less descriptive descriptors as discussed in Section 2.1.6. The same behaviour causes the number of matches to be relatively high compared to the number of keypoints (Figure 4.11).

The "AAU forest" and "Vicon Room 2" datasets have low numbers of inliers, which might impact robustness in such environments. The "AAU forest" is a relatively small, difficult environment for mapping containing noise and mapping errors. Only a few regions in maps are suitable for producing reliable matching points, which leads to the small number of inliers.

For the "Vicon Room 2" dataset, colour-based descriptors exhibited generally better performance (Figure 4.12) and they allow robust estimation for this dataset. The "Vicon Room 2" maps also contain noise caused by the dynamic motion of the aerial vehicle.

Interestingly in both cases with a limited number of inliers, the map-merging algorithm has been able to produce an accurate merge (Figure 4.10), even before ICP refining. The inliers proved to be the correct matches.

4.6 Descriptors

As I already noted in the previous section, the performance of the descriptors varies significantly in some cases (Figure 4.12). The PFHRGB descriptors exhibited the highest number of inliers across the datasets (Figure 4.12). The SHOT descriptors with colour also exhibited a decent amount of inliers, especially considering the fast processing speed (Section 4.7).

The lowest number of inliers have been produced by SC3D descriptors. Moreover, the PCL implementation of SC3D crashed in two instances. Together with RSD descriptors, which has not been able to produce any inliers and are not shown in the plots, the SC3D may not be suitable for map-merging in the evaluated setup.

Euclidean error distances (Figure 4.13) mostly reflects the number of inliers for respective descriptors, with PFHRGB descriptors generally providing the best initial transformation. With one exception, the ICP refining was able to produce high-quality alignment for all descriptors with small error differences, neglecting the differences between initial alignments for different descriptors. For the FPFH descriptors, the initial estimate in "Machine Hall" dataset has a significantly higher error and the ICP refining stuck in a local extreme.

4.7 Runtime performance

Although the runtime performance was not a primary concern of the presented algorithm, it is interesting to take a look at the processing time. In the default configuration (Figure 4.14), there are orders of magnitude differences between algorithm steps as presented in Section 2.1. Most of the processing time is taken by the SIFT keypoints detection and the computation of PFH descriptors.

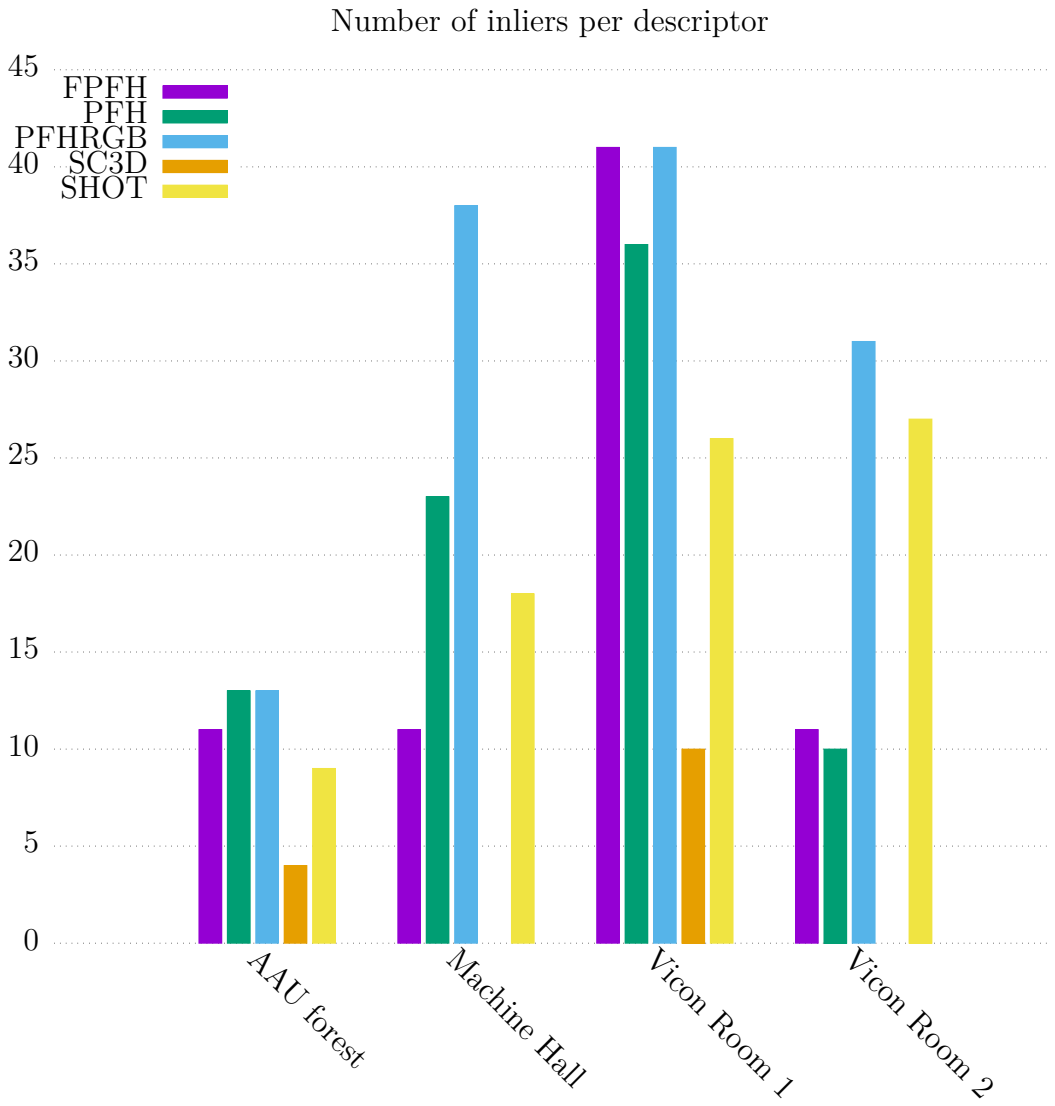


Figure 4.12: Number of inliers across the datasets for different descriptor algorithms. Apart from descriptors algorithms, the default parameters have been used including SIFT keypoints and the reciprocal matching algorithm (Section 2.1.6). The PCL implementation of SC3D descriptors crashed on "Machine Hall" and "Vicon Room 2" datasets. This might indicate that the implementation in PCL is probably not as mature as the remaining descriptors, but I have not investigated further.

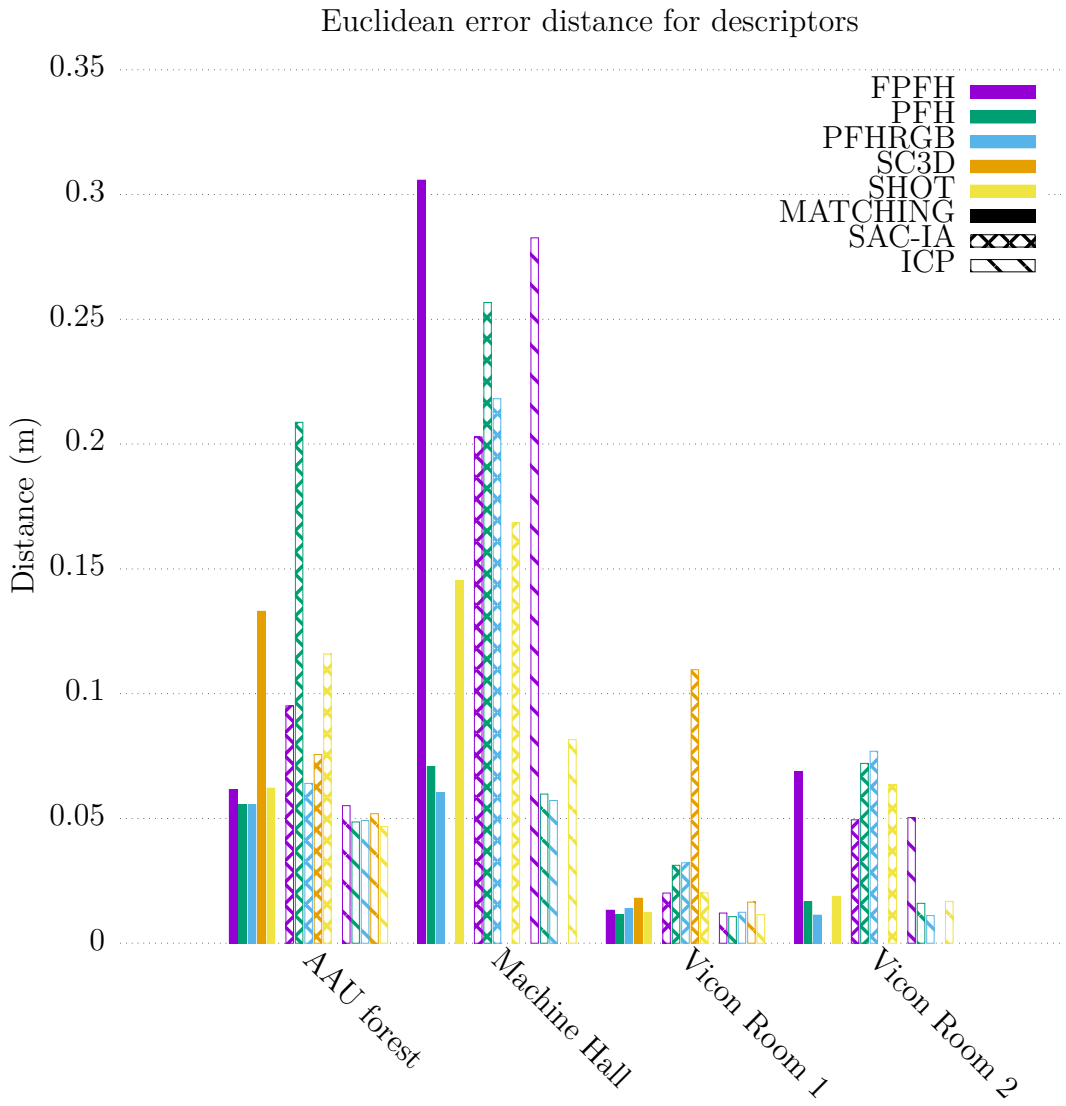


Figure 4.13: Euclidean error scores (the sum of Euclidean distances for corresponding points) across the datasets for different descriptor algorithms. Similar to Figure 4.10, the first two scores are for the initial estimates, using the reciprocal matching algorithm and the SAC-IA algorithm respectively, the last score is for the final refining with ICP, using the initial estimate from the reciprocal matching algorithm. The reciprocal matching algorithm uses its default configuration of $k = 5$ neighbours considered for matching.

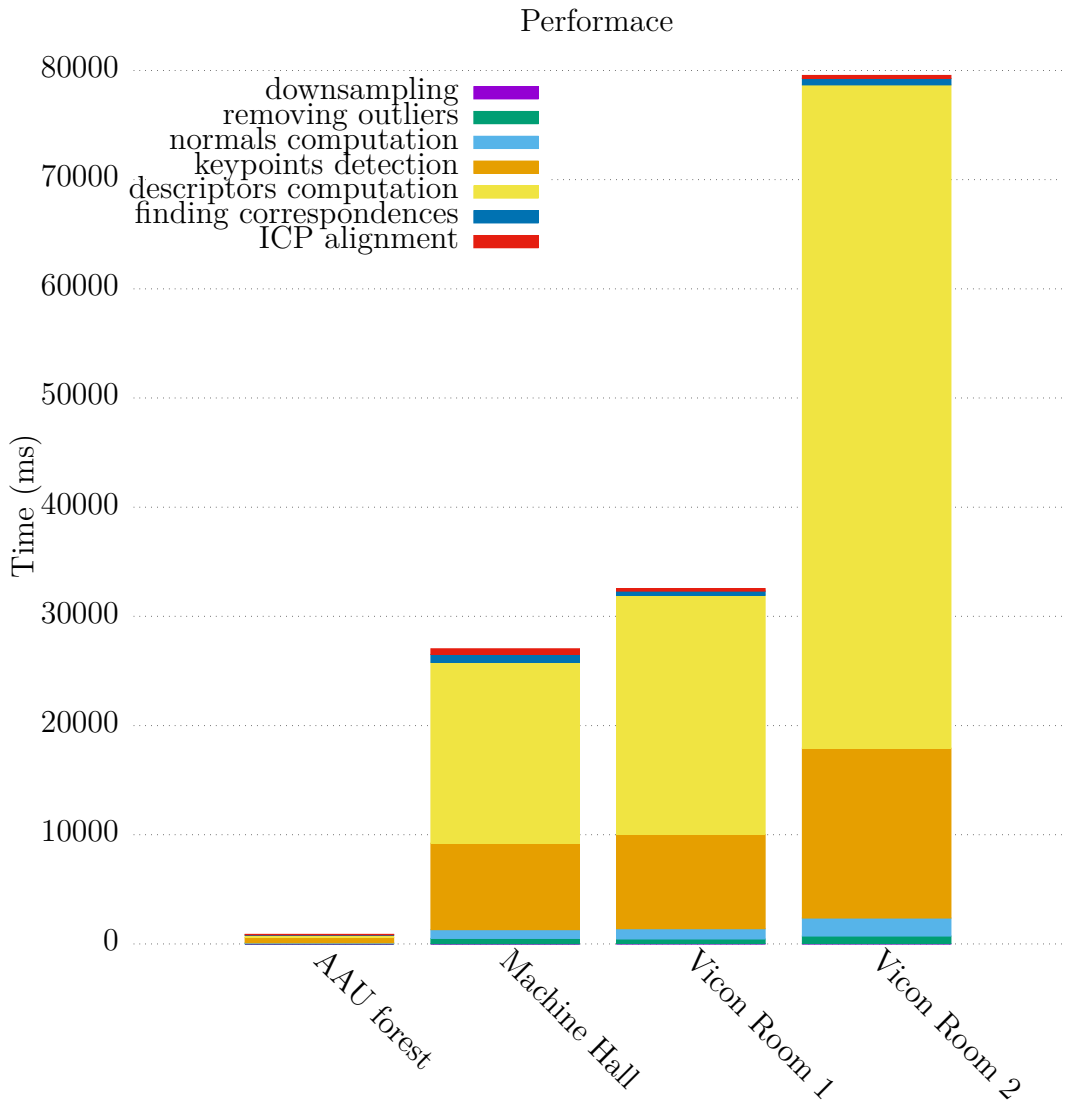


Figure 4.14: Processing time for the respective parts of the map-merging algorithm (Section 2.1) across the datasets. The data has been recorded during one run of the algorithm on Intel(R) Core(TM) i5-2520M and do not incorporate measurements from multiple runs to deal with various sources of non-determinism, although the runtime of the algorithm has been observed to be quite stable. The data are intended to show the order of magnitude differences of processing time between particular parts of the algorithm. The map-merging algorithm has been run with default parameters including PFH descriptors and SIFT keypoints.

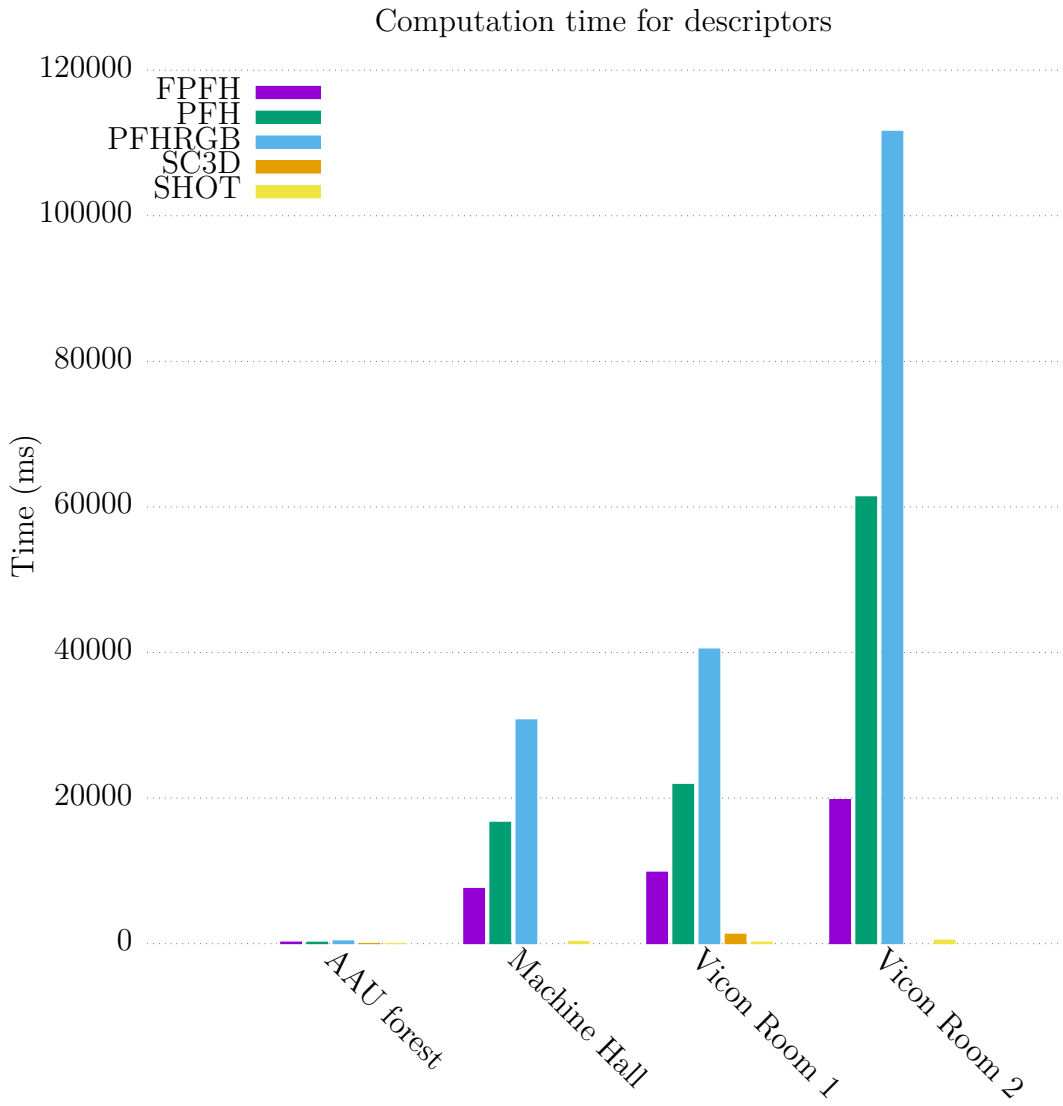


Figure 4.15: Processing time for descriptor algorithms across the datasets. Similar to Figure 4.14, the data has been recorded during one run of the algorithm on Intel(R) Core(TM) i5-2520M and are intended to show the order of magnitude differences of processing time between particular descriptors.

There are, however, significant differences between descriptors regarding processing time (Figure 4.15). All PFH-family descriptors require orders of magnitude higher processing time. There are also significant differences between the PFH-family, the processing time nearly doubles between FPFH, PFH and PFHRGB descriptors.

The SHOT descriptors with colour are very fast, which makes them a compelling choice, considering they also showed a good robustness (Figure 4.12). PFHRGB descriptors exhibited the best robustness, but also the longest processing time.

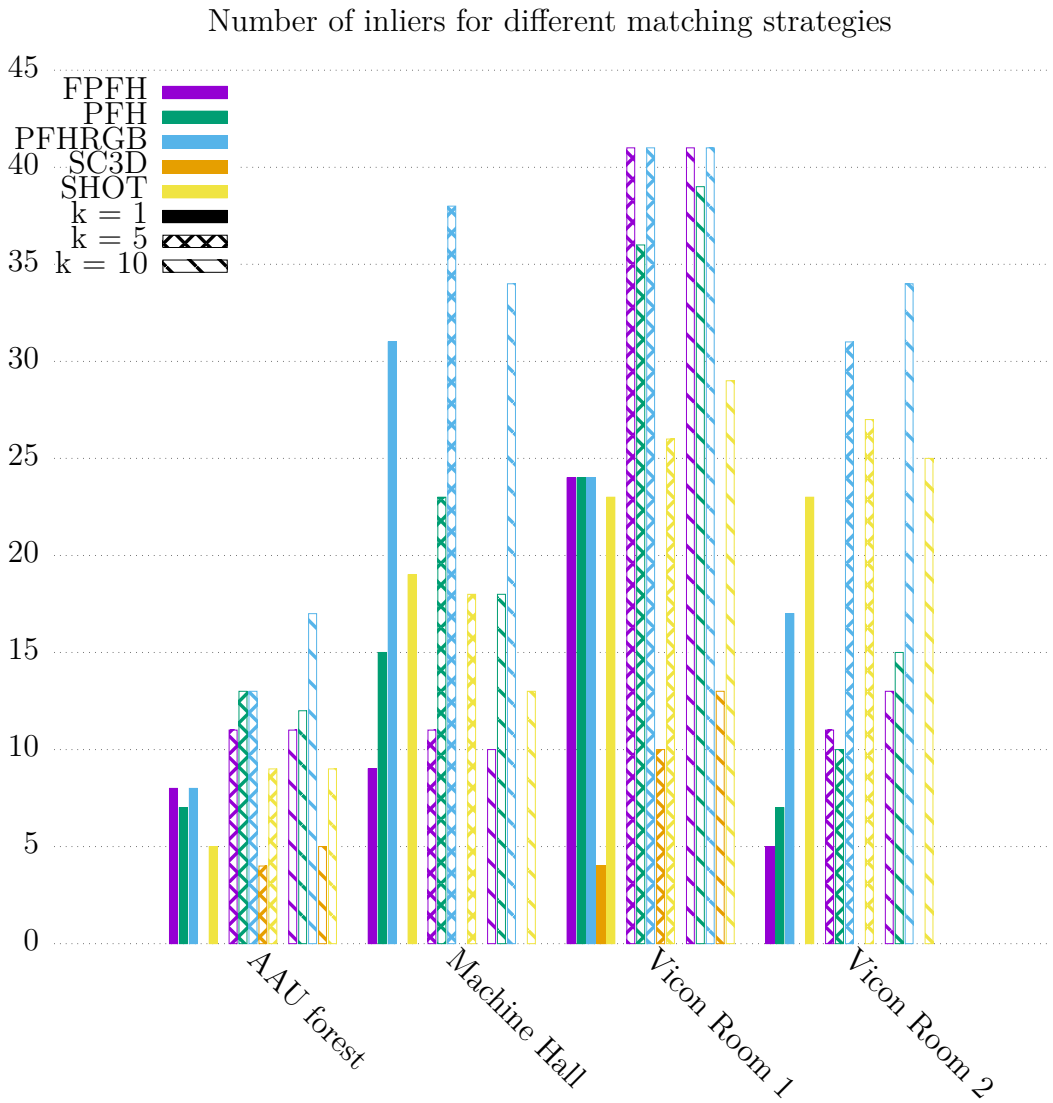


Figure 4.16: The number of inliers in matches produced by the presented reciprocal matching algorithm (Algorithm 3) for different k and the evaluated descriptors. For $k = 1$ the algorithm is the standard "1-on-1" matching with reciprocal match validation. The evaluation was running with default parameters.

4.8 Initial estimate algorithms

Comparing the presented reciprocal matching algorithm (Algorithm 3) with the SAC-IA algorithm, the reciprocal matching algorithm produces a better initial estimate in most of the evaluated cases (Figure 4.13). A significantly worse initial estimate was produced only for the FPFH descriptors in "Machine Hall" dataset, which may not come as a surprise as the SAC-IA algorithm was introduced with FPFH descriptors.

For PFH, PFHRGB and SHOT descriptors the reciprocal matching algorithm produced better initial estimates in the evaluation than the SAC-IA algorithm. Especially for the PFH and PFHRGB descriptors, the Euclidean error distance is significantly lower for the initial estimates produced by the reciprocal matching algorithm.

Comparing the presented algorithm with the standard "1-on-1" matching with

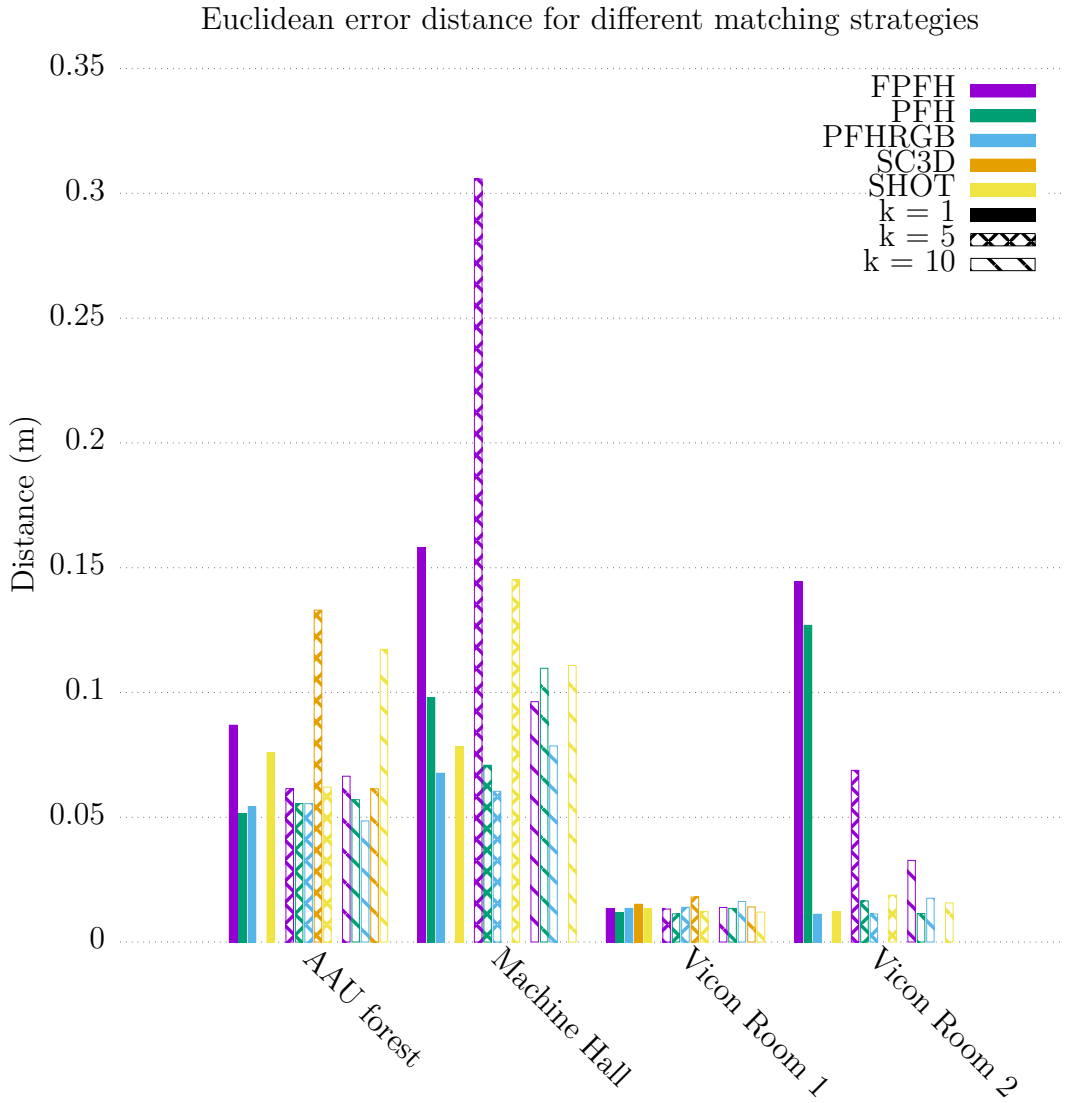


Figure 4.17: The Euclidean error distance for a different number of neighbours (k) in the presented reciprocal matching algorithm (Algorithm 3). For $k = 1$ the algorithm is the standard "1-on-1" matching with reciprocal match validation. The evaluation was running with default parameters.

reciprocal match validation, the presented approach can improve robustness in terms of number of inliers (Figure 4.16) and consecutively also accuracy in terms of smaller Euclidean distance (Figure 4.17). The standard "1-on-1" matching, which considers only the nearest match, is equivalent to the presented reciprocal matching with $k = 1$.

As expected, the increased number of neighbours considered for matching is beneficial for FPFH descriptors, where the increased k is essential for a good initial estimate in some cases. With higher than default $k = 10$, the reciprocal matching algorithm can produce a significantly better initial matching estimate than the SAC-IA algorithm.

Surprisingly, we can obverse the increased number of inliers also for PFH and PFHRGB descriptors with $k = 5$ (Figure 4.16). Broadening the neighbour search further to $k = 10$ does not seem to increase robustness significantly for PFH and PFHRGB descriptors, in some cases the number of inliers is even lower than for $k = 5$. The SHOT descriptors with colour generally slightly benefit from the increased k, but in the case of "Machine Hall" dataset, the number of inliers decreases with increasing k. For SC3D descriptors, there are fewer data as the implementation in PCL crashed in "Machine Hall" and "Vicon Room 2" datasets. However, it seems that SC3D descriptors work well with the increased k, in "AAU forest" dataset there are no inliers for the standard "1-on-1" matching, but the presented matching approach was able to generate 5 inliers with the increased k, yielding a good initial estimate (Figure 4.17).

Good performance across different descriptors makes the presented reciprocal matching a preferred matching strategy over the SAC-IA algorithm for the map-merging of point cloud maps.

4.9 Overlapping areas

As the presented map-merging algorithm (Chapter 2) relies entirely on information contained in the point cloud maps, properties of the overlapping area in maps (which is the only source of information for the map-merging) greatly influence the map-merging process.

First of all, the size of the common area in maps is the most influencing factor. Large common areas may contain more features, which in turn can yield more feature matches and inliers providing a more robust transformation estimate.

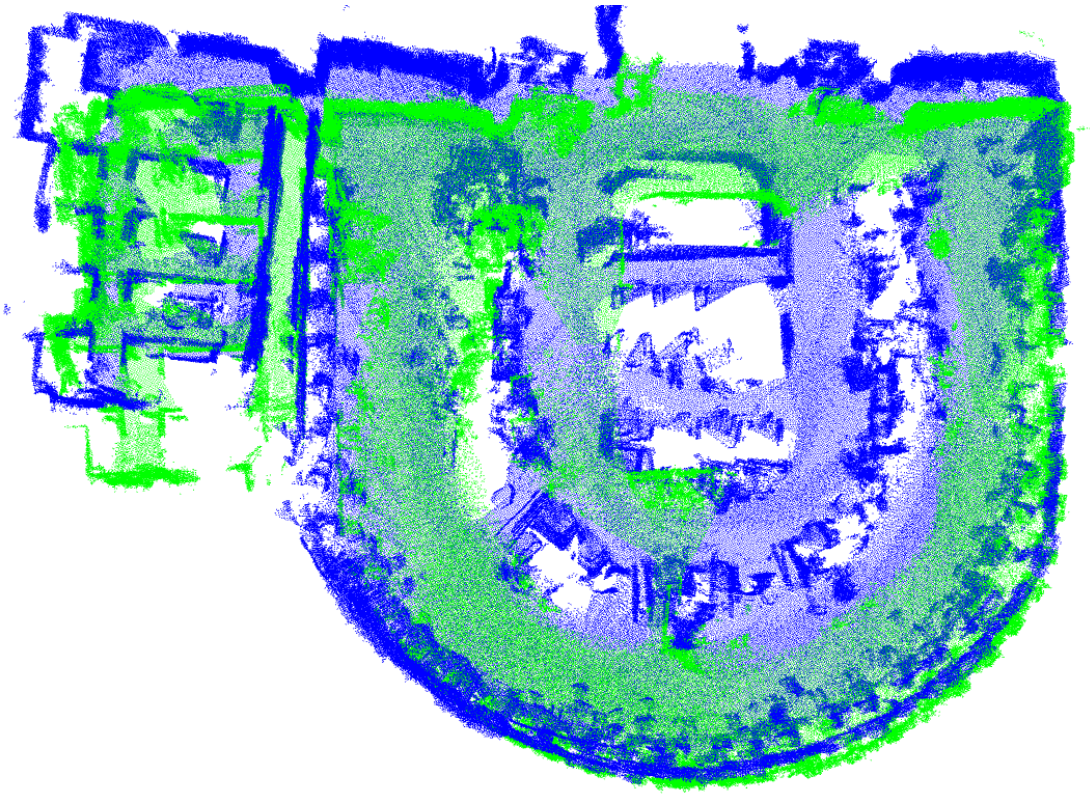
In real-world environments, however, even fairly large common areas might not be sufficient if they do not contain enough outstanding landmarks for the feature detection. These feature-poor areas are fairly common in man-made environments including industrial areas. Flat, single-colour surfaces typically pose a great challenge for common feature-based methods.

For example, "MFF Refectory" dataset (Figure 4.9) contains two maps with an overlapping area at the corridor. Although the area is fairly large, it has proven to be insufficient for a reliable transformation estimate and the map-merging. The corridor does not contain enough significant landmarks. Moreover, the same lack of landmarks in corridor areas leads to mapping errors in the maps, that make map-merging even more challenging. If no common feature-rich areas are present in maps, the map-merging based exclusively on point cloud maps is expected to be a challenging task.

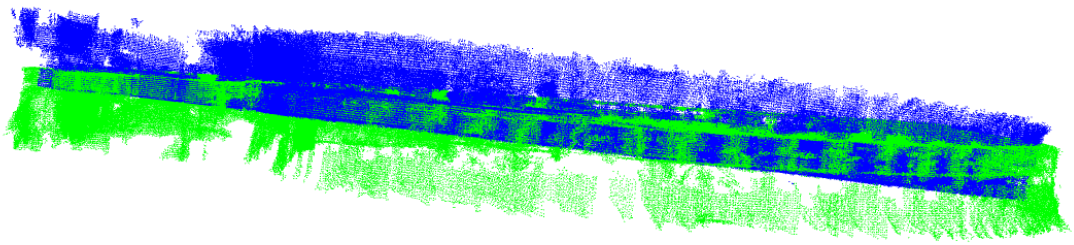
Ambiguities and self-similarities in the environment may also compromise the map-merging. "MFF Rotunda" dataset (Figure 4.8) was recorded in a symmetrical half-circular laboratory with many similar looking workplaces. The symmetry of the environment causes the SAC-IA algorithm to wrongly match the maps upside down (Figure 4.18).

The presented reciprocal matching algorithm (Algorithm 3) uses a stricter non-probabilistic approach and it is able to avoid the upside down match. However, inliers are clustered in one particular area of the map (Figure 4.19a), which leads to a visible angular error (Figure 4.19b). Other parts of the common area proved unable to provide stable matches, especially the circular area seems to contain more mapping errors and only unstable features.

This example shows that even a fairly large overlapping areas might not be suitable for the map-merging. Suitable common areas allowing a robust transformation estimate between maps should contain a decent amount of significant landmarks and should contain a minimal amount of ambiguities, self-similarities and symmetries.

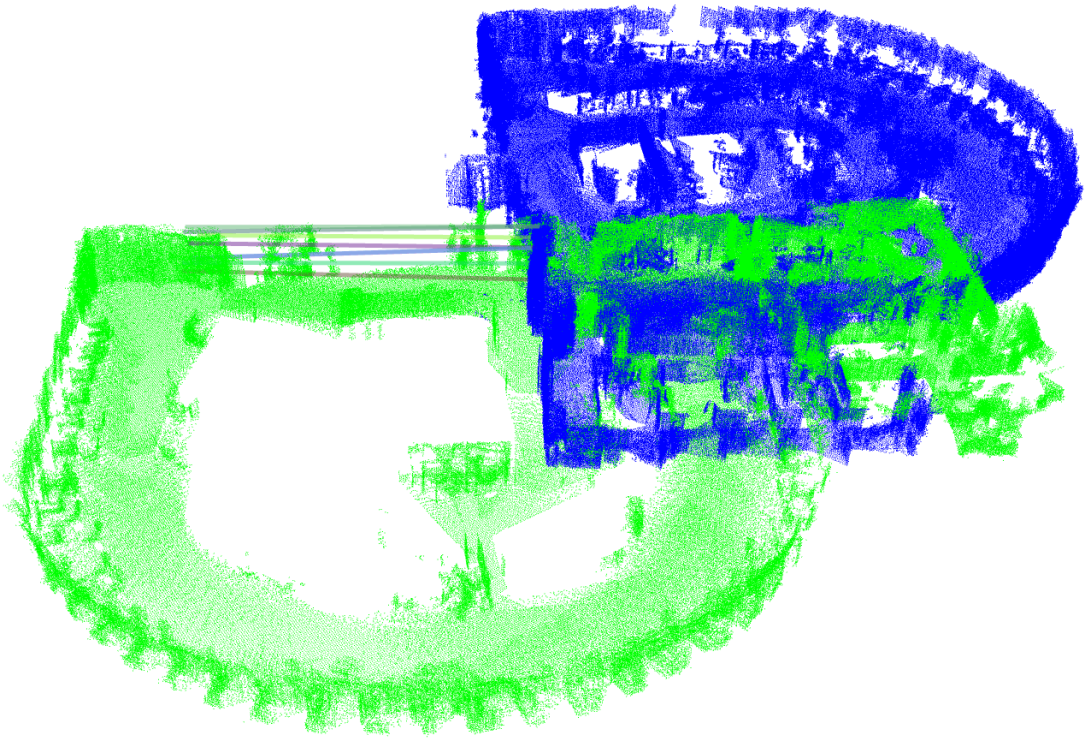


(a) Top view

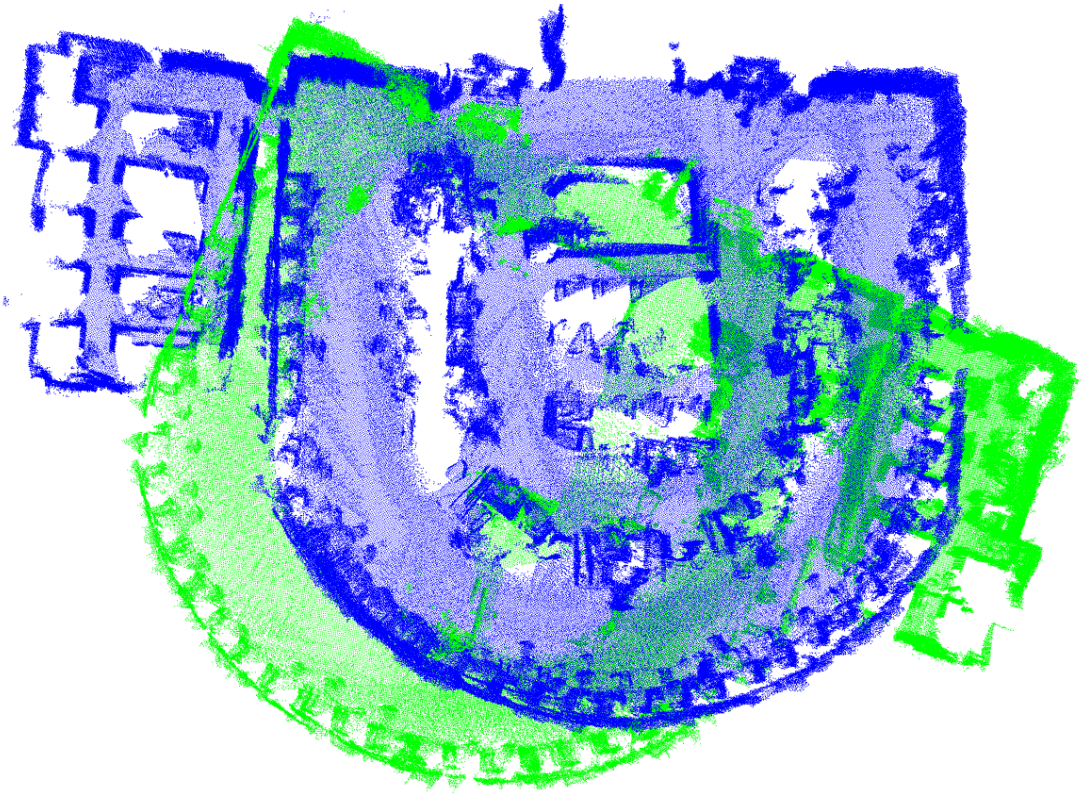


(b) Lateral view

Figure 4.18: The initial transformation estimate of the SAC-IA algorithm for "MFF Rotunda" dataset. The symmetry of the environment causes wrong upside down match of the second map.



(a) Inliers



(b) The initial estimate with an angular error

Figure 4.19: The initial transformation estimate and inliers of the presented reciprocal matching algorithm (Algorithm 3) for "MFF Rotunda" dataset. The inliers are clustered in a single area, which introduces an angular error in the initial transformation estimate.

Conclusion

This work presented a novel map-merging algorithm for merging 3D point cloud maps in multi-robot systems. The algorithm is based on feature-matching transformation estimation and works solely on point cloud maps without any additional auxiliary information. This makes the algorithm applicable in heterogeneous multi-robot systems and the algorithm can work with different SLAM approaches and sensor types. To the best of my knowledge, the presented approach is the first implemented map-merging algorithm working directly on point clouds without any extra information.

The work showed the feasibility of the feature-matching approach for registration of low-density point cloud maps produced by SLAM algorithms while using 3D point cloud features typically employed with high-density sensor data.

A reciprocal descriptor matching algorithm was introduced for estimating the initial transformation using feature-matching. The algorithm requires very little parametrisation and exhibited good performance across descriptors in the evaluation. In the most configurations, it outperforms SAC-IA algorithm for initial alignment available in the PCL library.

The map-merging algorithm has been evaluated on real-world datasets captured by both aerial and ground-based robots with a variety of stereo rig cameras and active RGB-D cameras. It has been evaluated in both indoor and outdoor environments ranging from forest to a single furnished room. The datasets used for evaluation include both well-established benchmark robotics datasets and my own experiments.

The proposed algorithm was implemented as a ROS package. To the best of my knowledge, it is the first ROS package for map-merging of 3D maps. The package has been submitted to the ROS distribution, the binary packages has been distributed with the ROS since the ROS Melodic Morenia release. The implementation does not require any particular communication solution between robots and can work with ROS in both multi-master and single-master setup. Likewise, the implementation does not presume any particular SLAM method, nor any particular sensor and uses a portable point cloud map representation, which makes it compatible with existing readily available SLAM implementations.

While the selected map representation enables great interoperability with existing software, the monolithic point clouds do not permit efficient repairing of mapping errors in the merged map. A pose graph of point clouds representation would be beneficial for the map-merging, but there is no standardised message format in the ROS for such a representation nor there is a common graph representation established across different SLAM implementations. In the future it would be beneficial to introduce a portable pose graph representation to the ROS, as discussed in Section 3.3.2, support it within the core ROS packages and promote its usage across SLAM implementations. This representation would allow the presented algorithm to work on sub-maps in the pose graph and repair the mapping errors in the merged map.

Bibliography

[1] Yasir Salih Osman Ali. *Development of Point Cloud Descriptors for Robust 3D Recognition*. PhD thesis, Universiti Teknologi Petronas, August 2015.

[2] Minoru Asada and Hiroaki Kitano. *RoboCup-98: Robot Soccer World Cup II*. Springer Science & Business Media, 1999.

[3] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Sensor Fusion IV: Control Paradigms and Data Structures*, volume 1611, pages 586–607. International Society for Optics and Photonics, 1992.

[4] Taigo Maria Bonanni, Bartolomeo Della Corte, and Giorgio Grisetti. 3-d map merging on pose graphs. *IEEE Robotics and Automation Letters*, 2(2): 1031–1038, 2017.

[5] Matthew Brown and David G Lowe. Automatic panoramic image stitching using invariant features. *International journal of computer vision*, 74(1): 59–73, 2007.

[6] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The EuRoC micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35(10):1157–1163, 2016.

[7] Stefano Carpin. Fast and accurate map merging for multi-robot systems. *Autonomous Robots*, 25(3):305–316, 2008.

[8] Howie Choset. Coverage for robotics–a survey of recent results. *Annals of mathematics and artificial intelligence*, 31(1-4):113–126, 2001.

[9] Bruno MF da Silva, Rodrigo S Xavier, Tiago P do Nascimento, and Luiz MG Gonsalves. Experimental evaluation of ROS compatible SLAM algorithms for RGB-D sensors. In *Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR), 2017 Latin American*, pages 1–6. IEEE, 2017.

[10] Alessandro Farinelli, Luca Iocchi, and Daniele Nardi. Multirobot systems: a classification focused on coordination. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(5):2015–2028, 2004.

[11] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[12] Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.

[13] Dieter Fox, Jonathan Ko, Kurt Konolige, Benson Limketkai, Dirk Schulz, and Benjamin Stewart. Distributed multirobot exploration and mapping. *Proceedings of the IEEE*, 94(7):1325–1339, 2006.

[14] Andrea Frome, Daniel Huber, Ravi Kolluri, Thomas Bülow, and Jitendra Malik. Recognizing objects in range data using regional point descriptors. In *European conference on computer vision*, pages 224–237. Springer, 2004.

[15] Dani Goldberg, Vincent Cicirello, M Bernardine Dias, Reid Simmons, Stephen Smith, Trey Smith, and Anthony Stentz. A distributed layered architecture for mobile robot coordination: Application to space exploration. In *In Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*. Citeseer, 2002.

[16] Gene H Golub and Christian Reinsch. Singular value decomposition and least squares solutions. *Numerische mathematik*, 14(5):403–420, 1970.

[17] Giorgio Grisetti, Cyrill Stachniss, Slawomir Grzonka, and Wolfram Burgard. A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In *Robotics: Science and Systems*, volume 3, page 9, 2007.

[18] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based SLAM. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.

[19] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[20] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2D LIDAR SLAM. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 1271–1278. IEEE, 2016.

[21] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013.

[22] Terry Huntsberger, Paolo Pirjanian, Ashitey Trebi-Ollennu, H Das Nayar, Hrand Aghazarian, Anthony J Ganino, Michael Garrett, Shirish S Joshi, and Paul S Schenker. Campout: A control architecture for tightly coupled coordination of multirobot systems for planetary surface exploration. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 33(5):550–559, 2003.

[23] Jiří Hörner. *Map-merging for multi-robot system*. Bachelor's thesis, Charles University, Faculty of Mathematics and Physics, Prague, 2016. URL `https://is.cuni.cz/webapps/zzp/detail/174125/`.

[24] Ilmir Z Ibragimov and Ilya M Afanasyev. Comparison of ROS-based visual SLAM methods in homogeneous indoor environment. In *Positioning, Navigation and Communications (WPNC), 2017 14th Workshop on*, pages 1–6. IEEE, 2017.

[25] Sergi Hernandez Juan and Fernando Herrero Cotarelo. Multi-master ROS systems. Technical Report IRI-TR-15-1, Institut de Robòtica i Informàtica Industrial, Universitat Politècnica de Catalunya, Llorens i Artigas 4-6, 08028, Barcelona, Spain, January 2015.

[26] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g2o: A general framework for graph optimization. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3607–3613. IEEE, 2011.

[27] Mathieu Labbé and François Michaud. Online global loop closure detection for large-scale multi-session graph-based SLAM. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 2661–2666. IEEE, 2014.

[28] Heon-Cheol Lee, Seung-Hwan Lee, Tae-Seok Lee, Doo-Jin Kim, and Beom-Hee Lee. A survey of map merging techniques for cooperative-SLAM. In *Ubiquitous Robots and Ambient Intelligence (URAI), 2012 9th International Conference on*, pages 285–287. IEEE, 2012.

[29] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[30] Zoltan-Csaba Marton, Dejan Pangercic, Nico Blodow, Jonathan Kleinehellefort, and Michael Beetz. General 3D modelling of novel objects from a single view. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 3700–3705. IEEE, 2010.

[31] Thomas Moore and Daniel Stouch. A generalized extended kalman filter implementation for the robot operating system. In *Intelligent Autonomous Systems 13*, pages 335–348. Springer, 2016.

[32] Jorge J Moré. The Levenberg-Marquardt algorithm: implementation and theory. In *Numerical analysis*, pages 105–116. Springer, 1978.

[33] Marius Muja and David G Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE transactions on pattern analysis and machine intelligence*, 36(11):2227–2240, 2014.

[34] Raul Mur-Artal and Juan D Tardós. ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[35] Lynne E Parker. The effect of heterogeneity in teams of 100+ mobile robots. *Multi-Robot Systems*, 2:205–215, 2003.

[36] Max Pfingsthorn, Bayu Slamet, and Arnoud Visser. A scalable hybrid multi-robot SLAM method for highly detailed maps. In *Robot Soccer World Cup*, pages 457–464. Springer, 2007.

[37] François Pomerleau, Francis Colas, Roland Siegwart, et al. A review of point cloud registration algorithms for mobile robotics. *Foundations and Trends® in Robotics*, 4(1):1–104, 2015.

[38] Radu Bogdan Rusu. *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, Computer Science department, Technische Universitaet Muenchen, Germany, October 2009.

[39] Radu Bogdan Rusu, Nico Blodow, Zoltan Csaba Marton, and Michael Beetz. Aligning point cloud views using persistent feature histograms. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 3384–3391. IEEE, 2008.

[40] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (FPFH) for 3D registration. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3212–3217. IEEE, 2009.

[41] Bastian Steder, Radu Bogdan Rusu, Kurt Konolige, and Wolfram Burgard. NARF: 3D range image features for object recognition. In *Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, volume 44, 2010.

[42] Federico Tombari, Samuele Salti, and Luigi Di Stefano. A combined texture-shape descriptor for enhanced 3D feature matching. In *Image Processing (ICIP), 2011 18th IEEE International Conference on*, pages 809–812. IEEE, 2011.

[43] Masahiro Tomono. Merging of 3D visual maps based on part-map retrieval and path consistency. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 5172–5179. IEEE, 2013.

[44] Xun S Zhou and Stergios I Roumeliotis. Multi-robot SLAM with unknown initial correspondence: The robot rendezvous case. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 1785–1792. IEEE, 2006.

List of Figures

List of Algorithms

List of Abbreviations

2D two-dimensional. 4, 6–8, 26, 35, 36

3D three-dimensional. 3, 4, 6–9, 11, 16, 25–27, 52, 63

AAU Alpen-Adria-Universität Klagenfurt. 13, 30, 33, 34

API Application Programming Interface. 64

BFS Breadth-First Search. 23

DFS Depth-First Search. 23

EKF Extended Kalman Filter. 36

FPFH Fast Point Feature Histogram. 16, 18, 41, 45, 46, 48, 66

GPS Global Positioning System. 6

ICP Iterative Closest Point. 9, 12, 19, 20, 36, 39, 41, 43, 66, 67

IMU Inertial Measurement Unit. 28–30, 35

NARF Normal Aligned Radial Feature. 9

PCL Point Cloud Library. 6, 21, 24, 27, 41, 42, 48, 52, 67, 68

PFH Point Feature Histogram. 9, 16, 18, 36, 39–41, 44–46, 48, 66

PFHRGB Point Feature Histogram with colour. 16, 18, 41, 45, 46, 48, 66

RANSAC Random Sample Consensus. 16–19, 21, 24, 36, 66

RGB Red-Green-Blue. 4, 11, 15

RGB-D Red-Green-Blue-Depth. 4, 9, 15, 28, 30, 35, 52

ROS Robot Operating System. 3, 6, 7, 24–28, 30, 36, 52, 60, 61, 63–65, 67

RSD Radius-based Surface Descriptor. 16, 41, 66

SAC-IA Sample Consensus Initial Alignment. 16–19, 21, 36, 39, 43, 46, 48–50, 52, 57, 58

SC3D 3D Shape Context. 16, 41, 42, 48, 66

SHOT Signature of Histograms of Orientations. 16, 41, 45, 46, 48, 66

SIFT Scale-Invariant Feature Transform. 8, 11, 15, 39–42, 44, 66

SLAM Simultaneous Localization and Mapping. 4, 6–9, 13, 21, 22, 24–26, 28, 30, 35, 36, 52, 63

SVD Singular-Value Decomposition. 19

List of Attached Files

This is a list of files in the electronic attachment to this work. The attachment contains source code for the presented ROS package (Chapter 3), the package documentation and experimental data (Chapter 4). Source code for the ROS package is also available online (`https://github.com/hrnr/map-merge`). A reproduction of the package documentation is attached in the printed version (Appendix A).

- `attachments.zip`
  - `evaluation` ..... datasets used in evaluation
    - `README.md`
    - `AAU` ..... Section 4.2
      - `AAU forest 2.pcd`
      - `AAU forest 1.pcd`
    - `EuRoC` ..... Section 4.1
      - `Vicon Room 1 02.pcd`
      - `Vicon Room 1 01.pcd`
      - `Machine Hall 02.pcd`
      - `Machine Hall 01.pcd`
      - `Vicon Room 2 01.pcd`
      - `Vicon Room 2 02.pcd`
    - `MFF` ..... Section 4.3
      - `MFF Rotunda 1.pcd`
      - `MFF Refectory 1.pcd`
      - `MFF Refectory 2.pcd`
      - `MFF Rotunda 2.pcd`
  - `map-merge` ..... sources for the ROS package
    - `LICENSE`
    - `README.md`
    - `map_merge_3d` ..... Section 3.3
      - `rosdoc.yaml`
      - `CMakeLists.txt`
      - `package.xml`
      - `CHANGELOG.rst`
      - `src`
        - `visualise.cpp`
        - `graph.cpp`
        - `visualise.h`
        - `map_merging.cpp`
        - `dispatch_descriptors.h`
        - `registration_visualisation.cpp`
        - `map_merge_node.cpp`
        - `matching.cpp`
        - `map_merge_tool.cpp`
        - `graph.h`
        - `features.cpp`
      - `launch`
        - `map_merge_3d.rviz`

```
        ├── from_pcds.launch
        └── map_merge.launch
    ├── include
    │   └── map_merge_3d
    │       ├── typedefs.h
    │       ├── map_merging.h
    │       ├── matching.h
    │       ├── map_merge_node.h
    │       ├── features.h
    │       └── enum.h
    ├── test
    │   └── test_map_merging.cpp
    └── doc
        ├── mainpage.dox
        ├── architecture.svg
        ├── screenshot.jpg
        ├── wiki.txt
        └── architecture.dia
└── doc ..... generated code documentation for the ROS package
    ├── html
    └── manifest.yaml
```

Appendices

A. map_merge_3d

This is a reproduction of the text available online at `http://wiki.ros.org/map_merge_3d`. Although maintained as the wiki the current version of the text reproduced below has been written solely by the author.

A.1 Package Summary

Merging multiple 3D maps, represented as point clouds, without knowledge of initial positions of robots.

- Maintainer status: developed
- Maintainer: Jiri Horner <laeqten AT gmail DOT com>
- Author: Jiri Horner <laeqten AT gmail DOT com>
- License: BSD
- Source: git `https://github.com/hrnr/map-merge.git` (branch: melodic-devel)

A.2 Overview

This package provides a 3D global map for multiple robots and the respective transformations between robots. It merges robots' individual maps based on the overlapping space in the maps and requires no dependencies on a particular SLAM or communication between the robots.

The ROS node can merge maps from the arbitrary number of robots. It expects maps from individual robots as ROS topics and does not impose any particular messaging between robots. If your run multiple robots under the same ROS master then `map_merge_3d` may work for you out-of-the-box, this makes it easy to setup a simulation experiment.

In the multi-robot exploration scenario your robots probably run multiple ROS masters and you need to setup a communication link between robots. Common solution might be multimaster_fkie package. You need to provide maps from your robots on local topics (under the same master). Also if you want to distribute merged map and tf transformations back to robots your communication must take care of it.

A.3 Architecture

`map_merge_3d` finds robot maps automatically and new robots can be added to the system at any time. 3D maps are expected as `sensor_msgs/PointCloud2`, other map messages are not supported.

Recommended topics names for robot maps are `/robot1/map`, `/robot2/map` etc. However the names are configurable. All robots are expected to publish

Figure A.1: Visualisation of registration between 2 maps using a `map_merge_3d` package.

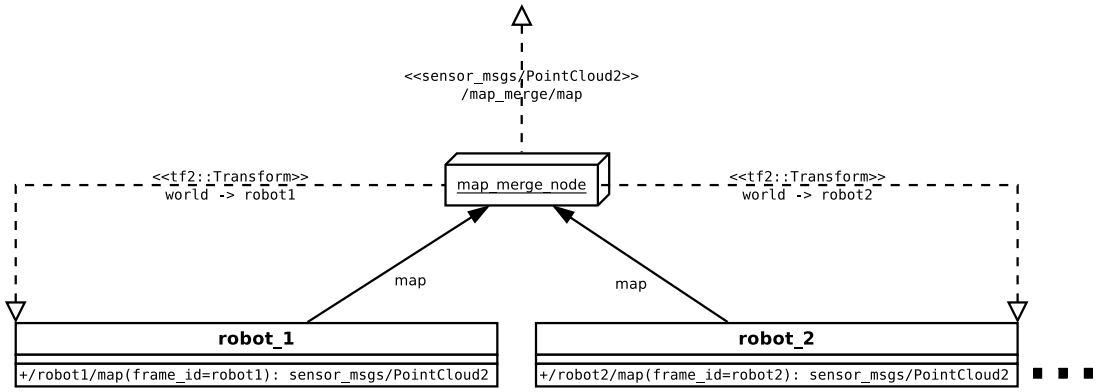


Figure A.2: Diagram showing ROS Application Programming Interface (API) of the map-merging node.

map under `<robot_namespace>/map`, where topic name (`map`) is configurable, but must be the same for all robots. For each robot `<robot_namespace>` is of cause different, but it does not need to follow any pattern. Further, you can exclude some topics using `robot_namespace` parameter, to avoid merging unrelated point clouds.

A.4 Estimation

Transformations between maps are estimated by feature-matching algorithm and therefore it is required to have sufficient amount of overlapping space between maps to make a high-probability match. If maps don't have enough overlapping space to make a solid match, merger might reject those matches.

Estimating transforms between maps is cpu-intesive so you might want to tune `estimation_rate` parameter to run the re-estimation less often.

A.5 ROS API

A.5.1 map_merge_node

Provides map merging services offered by this package. Dynamically looks for new robots in the system and merges their maps. Provides `tf` transforms.

Subscribed Topics

`<robot_namespace>/map` (sensor_msgs/PointCloud2)
Local map for a specific robot.

Published Topics

`map` (sensor_msgs/PointCloud2)
Merged map from all robots in the system.

Parameters

Node Parameters Parameters affecting general setup of the node.

`~robot_map_topic` (string, default: `map`)
Name of robot map topic without namespaces (last component of the topic name). Only topics with this name are considered when looking for new maps to merge. This topics may be subject to further filtering (see below).

`~robot_namespace` (string, default: `<empty string>`)
Fixed part of the robot map topic. You can employ this parameter to further limit which topics are considered during dynamic lookup for robots. Only topics which contain (anywhere) this string are considered for lookup. Unlike `robot_map_topic` you are not limited by namespace logic. Topics are filtered using text-based search. Therefore `robot_namespace` does not need to be a ROS namespace, but it can contain slashes etc. This string must be a common part of all maps topic name (all robots for which you want to merge map).

`~merged_map_topic` (string, default: `map`)

Topic name where merged map is published.

~`world_frame` (string, default: `world`)
Frame id (in tf tree) which is assigned to published merged map and used as reference frame for tf transforms.

~`compositing_rate` (double, default: `0.3`)
Rate in Hz. Basic frequency on which the node merges maps and publishes merged map. Increase this value if you want faster updates.

~`discovery_rate` (double, default: `0.05`)
Rate in Hz. Frequency on which this node discovers new robots (maps). Increase this value if you need more agile behaviour when adding new robots.

~`estimation_rate` (double, default: `0.01`)
Rate in Hz. Frequency on which this node re-estimates transformations between maps. Estimation is cpu-intensive, so you may wish to lower this value.

~`publish_tf` (bool, default: `true`)
Whether to publish estimated transforms in the tf tree. See below.

Registration Parameters Parameters affecting only registration algorithm used for estimating transformations between maps. These parameters should be defined in the same namespace as normal node parameters.

~`resolution` (double, default: `0.1`)
Resolution used for the registration. Small value increases registration time.

~`descriptor_radius` (double, default: `resolution * 8.0`)
Radius for descriptors computation.

~`outliers_min_neighbours` (int, default: `50`)
Minimum number of neighbours for a point to be kept in the map during outliers pruning.

~`normal_radius` (double, default: `resolution * 6.0`)
Radius used for estimating normals.

~`keypoint_type` (string, default: `SIFT`)
Type of keypoints used. Possible values are SIFT, HARRIS.

~`keypoint_threshold` (double, default: `5.0`)
Keypoints with lower response that this value are pruned. Lower this threshold when using Harris keypoints (you can set 0.0).

~`descriptor_type` (string, default: `PFH`)
Type of descriptors used. Possible values are PFH, PFHRGB, FPFH, RSD, SHOT, SC3D.

~`estimation_method` (string, default: `MATCHING`)
Type of descriptors matching algorithm used. This algorithm is used for initial global match. Possible values are `MATCHING`, `SAC_IA`.

~`refine_transform` (bool, default: `true`)
Whether to refine estimated transformation with ICP or not.

~`inlier_threshold` (double, default: `resolution * 5.0`)
Inlier threshold used in RANSAC during estimation.

~`max_correspondence_distance` (double, default: `inlier_threshold * 2.0`)
Maximum distance for matched points to be considered the same point.

~`max_iterations` (int, default: `500`)
Maximum iterations for RANSAC.

~`matching_k` (int, default: `5`)

Number of the nearest descriptors to consider for matching.

~`transform_epsilon` (double, default: `1e-2`)
The smallest change allowed until ICP convergence.

~`confidence_threshold` (double, default: `0.0`)
Minimum confidence in the pair-wise transform estimate to be included in the map-merging graph. Pair-wise transformations with lower confidence are not considered when computing global transforms. Increase this value if you are having problems with invalid transforms being estimated. The confidence value is computed as a reciprocal of Euclidean distance between transformed maps.

~`output_resolution` (double, default: `0.05`)
Resolution of the merged global map.

Provided tf Transforms

`world` → `mapX_frame`
Transformation from the world frame (which name can be configured using `world_frame` parameter) to each of the maps. Each map must have a correct `frame_id` set (instead `mapX_frame`) in the `sensor_msgs/PointCloud2` message. If the transformation could not be estimated, null transformation is published.

A.6 Tools

Alongside ROS node `map_merge_3d` provides command-line tools to work with point cloud maps saved in `pcd` files. Both tools accept any of the registration parameters described in Section A.5.1.

The tools use PCL command-line parsing module. PCL command-line parsing has some limits (PCL users won't be surprised): it supports only `--param value` format, `--param=value` is not accepted. Unknown options are ignored. Options may be arbitrarily mixed with filenames. There are no short versions for parameters.

A.6.1 map_merge_tool

Tool for merging maps offline. Produces `output.pcd` with merged global map. This tool can merge arbitrary number of maps.

Usage

```
rosrun map_merge_3d map_merge_tool [--param value] map1.pcd
map2.pcd [map3.pcd...]
```

For example to use SHOT descriptors with 3 maps:

```
rosrun map_merge_3d map_merge_tool --descriptor_type SHOT map1.pcd
map2.pcd map3.pcd
```

A.6.2 registration_visualisation

Visualises pair-wise transform estimation between 2 maps. Uses PCL visualiser for the visualisation.

Usage

```
rosrun map_merge_3d registration_visualisation [--param value]
map1.pcd map2.pcd
```

After one step of the estimation a visualisation window appears. You can freely navigate the point cloud, save a screenshot or camera parameters (press `h` to see all shortcuts). After the window is closed, estimation continues with the next phase and the next visualisation window appears. Details about estimation progress are printed to `stdout`.

A.7 Acknowledgements

This package was developed as part of my master thesis at Charles University in Prague.