Norges miljø- og
biovitenskapelige
universitet

Master's Thesis 2024 30 ECTS
REALTEK

Comparative study of SLAM systems in various environments using different sensing technologies

Sammenligningsstudie av SLAM-systemer i ulike miljøer ved bruk av forskjellige sensorteknologier

Tor Erik Aasestad & Marko Miric
Applied Robotics

Abstract

This thesis explores a comparative study on camera based and LIDAR based Simultaneous Localization and Mapping (SLAM) systems across diverse environments, specifically agricultural orchards and urban parking lots. For camera based systems, it also includes indoor environments.

Traditional evaluations of SLAM technologies often rely on well-known datasets such as KITTI and TUM, which may not entirely reflect the operational challenges encountered in different real-world scenarios. Our study extends these evaluations to include real-world datasets collected from specific environments using both a RGB-D camera and a LIDAR. This research utilizes a range of SLAM systems, including ORB-SLAM2, ManhattanSLAM, RESLAM, RGBDSLAMv2 and RTAB-Map for camera-based methods, as well as A-LOAM, F-LOAM, KISS-ICP, HDL Graph SLAM, SC-A-LOAM and LeGO-LOAM for LIDAR-based methods. We measured the accuracy of the pose estimation and the density of the resulting maps of these systems, aiming to identify which technologies perform best under different environmental conditions without adjusting the system parameters.

Initial findings indicate that environmental complexity significantly affects SLAM performance, with LIDAR-based systems showing greater robustness in less structured environments. This comprehensive comparison not only aids in understanding the relative strengths and weaknesses of SLAM systems in different contexts, but also guides future enhancements in SLAM technology tailored to specific environmental challenges.

Sammendrag

Denne oppgaven tar for seg en sammenligningsstudie mellom kamera baserte og LIDAR baserte Simultaneous Localization and Mapping (SLAM) systemer i forskjellige miljøer, spesifikt agrikulturelle frukthager og urbane parkeringsplasser. For kamera baserte systemer, blir også innendørs miljøer inkludert.

Tradisjonelle evalueringsmetoder av SLAM teknologier baserer seg ofte på kjente datasett som KITTI og TUM som ikke alltid representerer de utfordringene som man møter i virkelige scenarioer. Vår studie utvider disse evalueringene til å inkludere nye datasett som inneholder både et RGB-D kamera og en LIDAR sensor. Denne studien sammenligner en rekke SLAM systemer, som inkluderer ORB-SLAM2, ManhattanSLAM, RESLAM, RGBDSLAMv2 og RTAB-Map for kamera baserte metoder og A-LOAM, F-LOAM, KISS-ICP, HDL Graph SLAM, SC-A-LOAM og LeGO-LOAM for LIDAR baserte metoder. Vi måler nøyaktigheten til den estimerte posisjonen og punkttettheten i det resulterende kartet for hver av de forskjellige systemene med mål om å identifisere hvilket av sensorene som gir best resultater i forskjellige settinger uten å justere parametere.

Våre funn viser at komplekse miljøer påvirker SLAM systemene i stor grad og LIDAR baserte systemer viser mer robusthet i mindre strukturerte miljøer. Denne omfattende sammenligningen bidrar ikke bare til å forstå de relative styrker og svakheter med forskjellige SLAM-systemer i forskjellige sammenhenger, men også til å veilede fremtidige utviklinger i SLAM-teknologi tilpasset spesifikke miljøer.

Acknowledgements

We would like to express our gratitude to our supervisor, Weria Khaksar, for his guidance, useful feedback, good discussions and overall enthusiastic engagement and support towards this thesis. We would also like to thank Håvard Pedersen Brandal for his contributions on inputs for the thesis and for aiding in hardware configurations.

We also show appreciation to our co-students at Applied Robotics and teachers at the robotics group for their friendship, support and knowledge sharing throughout our studies.

Finally, we express our sincere gratitude to our friends and families for supporting us in our journey here at NMBU.

Tor Erik Aasestad & Marko Miric
Norwegian University of Life Sciences (NMBU)
May, 2024

Contents

List of Figures

List of Tables

List of Abbreviations

APE Absolute Pose Error

EMM Environment Measurement Model

GB Gigabyte

GNSS Global Navigation Satellite System

GUI Graphical User Interface

ICP Iterative Closest Point

IMU Inertial Measurement Unit

LIDAR Light Detection And Ranging

LOAM Lidar Odometry And Mapping

MB Megabyte

ORB Oriented FAST and Rotated BRIEF

RGB-D Red Green Blue-Depth

RMSE Root Mean Square Error

ROS Robot Operating System

RTK Real-time Kinematic Positioning

SIFT Scale-Invariant Feature Transform

SURF Speeded-Up Robust Features

SLAM Simultaneous Localization And Mapping

SSD Solid-State Drive

std Standard Deviation

VM Virtual Machine

WSL Windows Subsystem for Linux

w.r.t with respect to

1. INTRODUCTION

Simultaneous Localization And Mapping (SLAM) is a technology that is advancing and becoming increasingly more important by the day, particularly in the field of autonomous systems, including autonomous driving, drones, and other various robotics applications.

In the realm of autonomous vehicles, SLAM serves great purpose in navigation, where it allows accurate position calculation and mapping in real-time. It also pushes these vehicles towards achieving extraordinary levels of autonomy, efficiency, and safety. In the aerospace industry, drones utilize SLAM for enhanced navigation capabilities, revolutionizing tasks such as search and rescue operations, as well as inspection and surveillance.

Moreover, the impact that SLAM can have is not only limited to urban and industrial settings. In agriculture, this technology revolutionizes farming practices by contributing to innovative solutions, such as crop monitoring, precise field mapping and autonomous farming navigation. Thus, integrating SLAM systems in agriculture introduces the prospect of improving efficiency, resource utilization, and overall productivity.

1.1. Motivation

The rapid advancement and increasing deployment of autonomous systems in various sectors highlights the need of having robust and reliable navigational technologies. SLAM, as a cornerstone technology, has revolutionized how machines perceive and interact with their surroundings. However, the success of these systems heavily relies on their ability to operate under varied and unpredictable environmental conditions. Historically, the development and testing of SLAM technologies have been predominantly centered around the environments that can be easily controlled or predicted, such as some urban landscapes and indoor settings, using datasets like KITTI and TUM. These environments, while critical, represent only a fraction of the potential applications for SLAM technologies.

Thus, the motivation for this study stems from the recognition that the real world presents a broader array of challenges, which includes everything from fluctuating lighting conditions to dynamic obstacles in outdoor environments. This diversity often leads to performance discrepancies that are not well-documented in conventional SLAM studies. By extending the scope of testing to include non-traditional environments such as agricultural fields and other urban areas, this research aims to fill a gap in existing literature. This investigation is driven by the broader goal of enabling technological advancements that can seamlessly transition into the practical demands of our ever-evolving world.

1.2. Problem statement and objective

While there are multiple studies ([10], [11], [12]) that compare various SLAM approaches, often using popular datasets such as KITTI [13] and TUM [14], this thesis aims to extend the comparison to agricultural and other urban environments. It is well documented that SLAM technologies may exhibit optimized performance for the specific conditions and scenarios captured in these datasets. Recognizing this potential bias, our research seeks to investigate how camera-based and LIDAR-based SLAM technologies perform in varied settings, including orchards and parking lots, without changing the inherent parameters. This approach not only challenges the generalizability of current SLAM systems but also addresses the need for broader validation across diverse real-world applications.

To create an evaluation baseline, the camera-based and LIDAR-based methods are firstly ran on TUM [14] and KITTI [13], respectively. This provides a benchmark for further analysis with other datasets and confirms that the methods are implemented correctly. The SLAM systems will then be evaluated on the self collected dataset to assess the performance of each of the sensor-based SLAM systems within the same environments. To add to this, the parameters of the SLAM systems will not be altered for the TUM and KITTI dataset, nor the collected dataset, as mentioned earlier. This is done to maintain consistency and fairness in the comparative analysis between different environments.

1.3. Main contribution

In contributing to methodological diversity using various datasets, this thesis aims to highlight the importance of testing SLAM technologies in a wider range of environments beyond those typically represented in benchmark datasets. By doing so, it not only provides a more comprehensive understanding of SLAM performance across different domains but also encourages the development of systems that are more adaptable and effective, regardless of the application context. This endeavor aims to inspire other researchers to consider a broader spectrum of testing environments and scenarios when developing future SLAM frameworks. This could potentially contribute to innovations in SLAM technologies that can transition from urban landscapes to more complex environments, such as agricultural fields, without fine tuning the parameters in between each environment. Such a methodological expansion is crucial for advancing the field toward solutions that are robust, versatile, and capable of meeting the challenges of an increasingly automated and diverse world.

2. THEORY

2.1. Simultaneous Localization and Mapping

SLAM is briefly and well explained by Xiang Gao and Tao Zhang from the book "Introduction to Visual SLAM" [1]:

"Simultaneous Localization and Mapping usually refer to a robot or a moving rigid body, equipped with a specific sensor, that estimates its motion and builds a model of the surrounding environment, without a priori information"

In more formal terms, the SLAM problem can be defined as finding the posterior probability p over the path $\mathbf{x}_{1:t}$ and map m given the data $z_{1:t}$ and control signals $u_{1:t}$:

$$p(\mathbf{x}_{1:t}, m | z_{1:t}, u_{1:t}) \tag{1}$$

The vehicle's path $\mathbf{x}_{1:t}$ is a sequence of poses $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t$, where each pose $\mathbf{x}_i$ typically includes the vehicle's position and orientation. The data $z_{1:t}$ represents cumulative sensory observations, while $u_{1:t}$ denotes control inputs that influence the vehicle's trajectory [15].

The difficulty of SLAM arises from the need to manage the uncertainty inherent in both the map and the vehicle's location. A key aspect of addressing the SLAM problem is the formulation of a motion model, which relates the current pose $\mathbf{x}_t$ to the previous pose $\mathbf{x}_{t-1}$ and the control input u_t:

$$p(\mathbf{x}_t | \mathbf{x}_{t-1}, u_t) \tag{2}$$

Thus, the motion model acts as a prediction step since it predicts the robot's new pose based on the previous pose and control input. This prediction serves as a starting point for the SLAM algorithm to incorporate new sensor information and to update its belief about the robot's location and map.

Additionally, the measurement model correlates sensory observations z_t with the map m and the current pose $\mathbf{x}_t$:

$$p(z_t | \mathbf{x}_t, m) \tag{3}$$

The measurement model plays a critical role in the SLAM process by linking the sensory data to the estimated map and the robot's pose. This model is integral for updating the belief about the robot's position and refining the map based on new observations. It effectively filters noisy sensor data to improve estimation accuracy [15].

2.1.1. Map representations

Within robotics there are multiple approaches on how to represent the environment using maps, this section will aim to introduce the specific representation that is applied by the SLAM systems used in this thesis.

Feature-based maps

Feature-based mapping is a technique that enables the construction of maps using a combination of detection, extraction and usage of distinct environmental features. These features include edges, corners, or identifiable objects and are used to create a spatial representation of the surroundings. Algorithms such as SIFT [16], SURF [17] and ORB [18] are utilized to identify and describe these features effectively. Once identified, the spatial relationships between these matched features are analyzed using techniques such as RANSAC [19] for robust estimation, which helps align the scans accurately, even amid noise and outliers. The result is a feature-based map that represents the environment through key features, each characterized by its own unique descriptor. This form of mapping not only aids in efficient localization and navigation, but also reduces the computational load compared to using dense point cloud maps.

Point cloud maps

Point cloud maps is a popular map representation used in SLAM and it is the result of one or more point clouds being combined into one, unified map, using techniques such as ICP [20]. Each point is assigned a label with x, y, z coordinates. The point clouds are especially useful for visualizing the environment, which helps identifying and evaluating the performance of mapping methods. However, despite being useful for visualization, point clouds present significant challenges in terms of data management and processing. The large number of data points require substantial computational resources for storage, processing and real-time analysis. To alleviate computational cost, there are processing techniques such as filtering [21] and downsampling [22] that may help in refining the point cloud by removing noise. An example of how a point cloud looks like before and after downsampling is shown in Fig. 1. By applying these techniques to the point cloud map, it may aid in reducing data size and computational costs while improving localization and mapping processes.

Volumetric maps

Volumetric maps are a form of occupancy grid in 3D. One common example of this is an octomap. To increase the efficiency, the authors proposed to have a dynamic resolution, where free cells can have a lower resolution than the occupied cells [24]. A visualization of this dynamic size can be seen in Fig. 2.

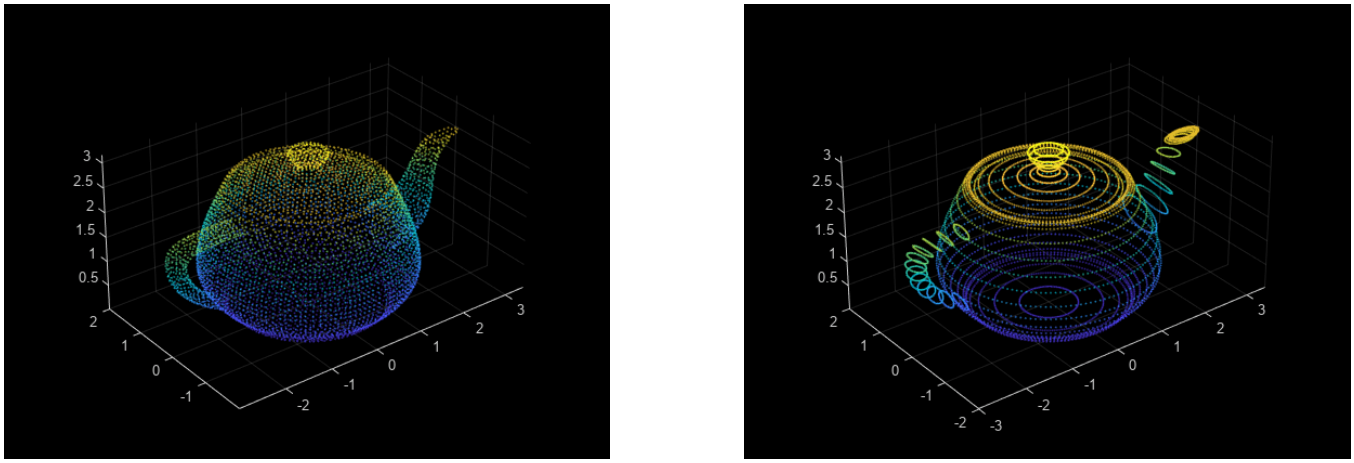


Fig. 1: Example of downsampling of point cloud. The left figure is the original figure while the right image shows the downsampled version [23].

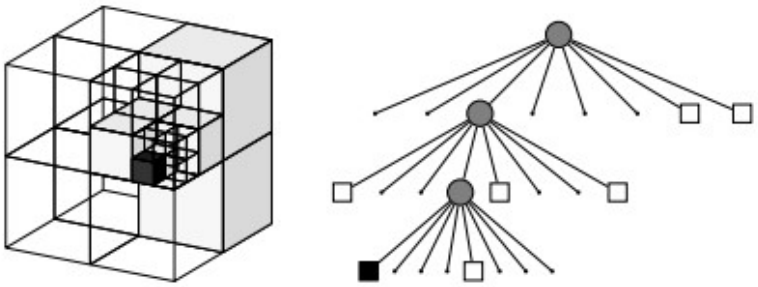


Fig. 2: An example of how the octotree is represented in 3D where the free blocks are white and the black cells represent the occupied cells. This image is from the original octomap paper [24].

2.1.2. Graph-based SLAM

To overcome the complexity of map representation, a popular approach is to use a graph to represent the map. The graph allows for easy access to each node in the graph and allows for efficient minimization of the errors as the robot moves around in its environment.

The graph is created by creating a node for each pose during the mapping process. These can be obtained from the odometry of the wheels or other sensors which allows for capture of poses from the robot. The edges of the graph represent the constraints that describe the relationship of the nodes. This can be information such as odometry information at the two different nodes or sensor information from each of the poses. To optimize the configuration of the nodes and edges, most optimization techniques can be applied. Including, but not limited to, Gauss-Newton and least-squares error minimization [25].

If the robot returns to a previously explored area after traveling in an unexplored area, the algorithm checks the nearby nodes to detect whether the older scan matches the new scan. If the scan matches, then it adds edges from the current node to the nodes where the scan matches. This new edge contains information

about the transformation that allows the best overlap in the scans [25]. Fig. 3 shows how a 2D graph-based SLAM can be represented where the poses x_j are in chronological order from 1 to t.

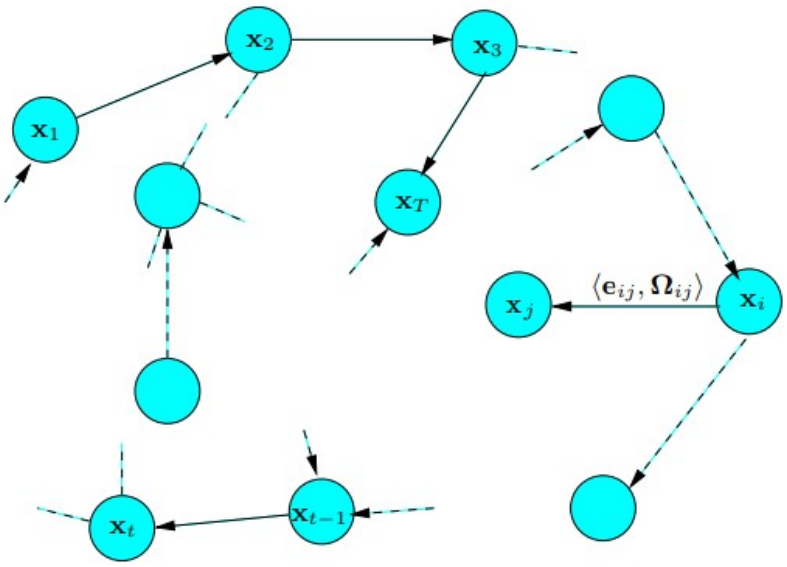


Fig. 3: Graph-based SLAM representation, where the x represent the poses and the edges are represented by the dotted lines between nearby nodes [25].

2.1.3. Feature-based SLAM

In feature-based SLAM, the main focus lies on identifying areas in an image that have unique characteristics. These features vary in complexity; they can either be simple geometric shapes, super-pixels, or even semantically labeled objects. The key attribute of a good feature is its repeatability, which allows it to be recognized consistently across various framed captured from different perspectives [26].

One crucial element of feature-based SLAM, is the ability to extract features and match them in various lighting conditions and viewpoints, as well as being computationally efficient to track. This necessitates the use of advanced algorithms for feature detection such as SIFT [16], SURF [17], or ORB [18], as noted earlier, which provides a balance between accuracy and processing speed.

In terms of scalability, while feature-based approaches are less memory-intensive compared to other SLAM methods, they may struggle in environments that have few distinct features, such as hallways or blank walls. In addition to this, feature-based SLAM have challenges related to dynamic environments, where moving objects could obstruct the consistency of feature detection [27]. However, there are ways to tackle this; by introducing additional sensors or using semantic labeling in environments that are not feature-rich, as well as incorporating techniques such as dynamic object recognition and tracking to enhance the robustness under these conditions.

2.2. Visual SLAM approaches

There are various approaches that are used in visual SLAM, and they usually share the same underlying structure as seen in the classical framework for visual SLAM, showcased in Fig. 4. However, there are slight distinctions between the various approaches that researchers have implemented for each unique SLAM system. This offers a large quantity of applications each of them can be used on, where some of them may be more suited in certain environments than others.

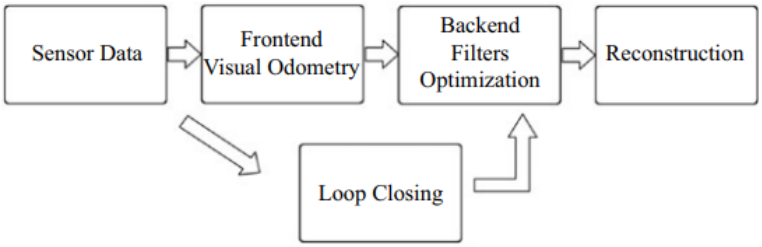


Fig. 4: Classic visual SLAM framework [1]

2.2.1. ORB-SLAM2

ORB-SLAM2 [2] is an open-source SLAM system that works with monocular, stereo, and RGB-D cameras. It is a continued version of the original ORB-SLAM [28] which only worked with monocular cameras. While somewhat old, around 7 years at the time of writing, it is still considered state-of-the-art and has inspired other SLAM algorithms that preceded it.

ORB-SLAM2 has three parallel threads it operates in. These include tracking, local mapping, and loop closure, as shown in Fig. 5. First up is the tracking thread, which locates the sensor by finding the corresponding features while simultaneously minimizing the reprojection error. After this, the map management operations are done by the local mapping thread. And at last, the loop closing, is responsible for the detection of new loops and correcting the drift error in the loop, and as stated earlier, some of the key concepts from the classical visual SLAM framework is clearly utilized here. When the last iteration of the three threads are finished, the algorithm further enhances the structure, while also ensuring motion consistency by performing a full bundle adjustment.

Due to ORB-SLAM2's sophisticated back-end, which is based on bundle adjustment, it can create high accuracy trajectory estimations and map reconstructions. Furthermore, the system features advanced map management capabilities, including the ability to reuse maps, perform loop closing, and relocalize within a known map [2]. This may contribute to providing accurate localization and robust mapping even in large and complex environments.

As ORB-SLAM2 is very reliant on good quality visual features for operation, it implies that the system

may struggle in low-texture environments where feature detection is difficult or in environments where there is either too much or too little light [29].

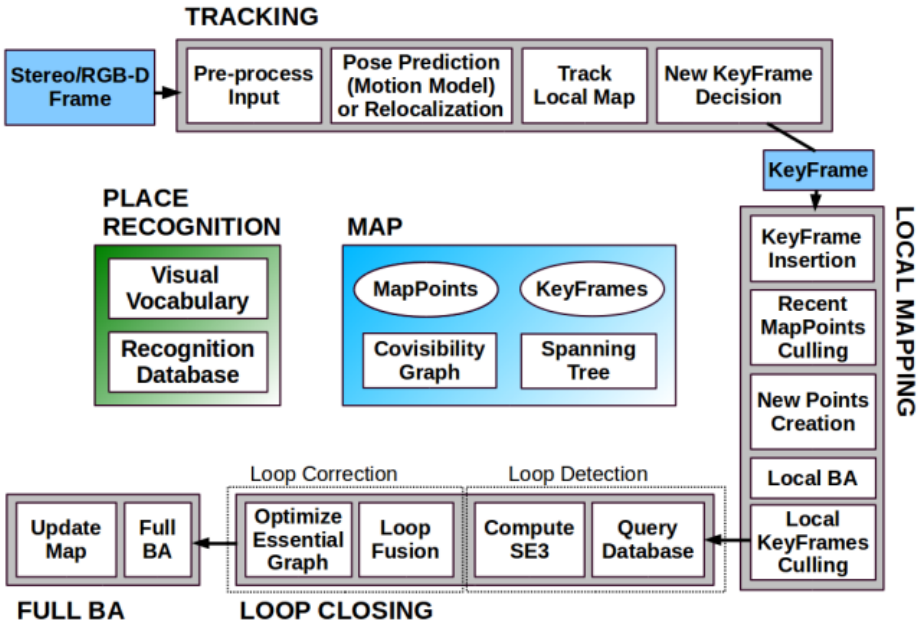


Fig. 5: ORB-SLAM2 framework structure [2]

2.2.2. RGBDSLAMv2

RGBDSLAMv2 [3], which operates with ROS, builds upon the foundations laid by earlier SLAM systems by incorporating more sophisticated algorithms and models for feature detection, matching, and optimization. Its main advantage, is its incorporation of an Iterative Closest Point (ICP) algorithm as well as an Environment Measurement Model (EMM) to validate the estimated transformations. These improvements allow for better performance in terms of accuracy, speed, and robustness to diverse environments and lighting conditions.

The system implements a graph-based approach. Graph optimization techniques are used to refine the pose graph, ensuring that the map remains consistent as new data is processed. This helps to correct any drift that may occur over time, improving the overall representation of the generated map. The framework structure is shown in Fig. 6.

Despite this, the computational demand of RGBDSLAMv2 is quite resource-intensive w.r.t real-time performance. This is mainly due to its use of SIFT features, which are much slower than e.g. ORB features. Moreover, it is highly reliant on slow movement of the sensor to obtain good tracking [30].

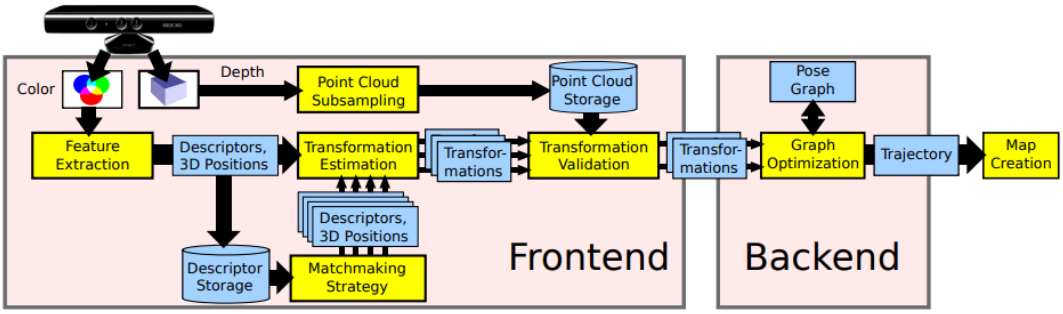


Fig. 6: RGBDSLAMv2 framework structure [3]

2.2.3. RTAB-Map

RTAB-Map [4] is another graph-based SLAM approach, which supports integration with ROS as the rtabmap_ros package since 2013. It operates by receiving odometry as an external input, allowing flexibility in the choice of odometry sources suitable for various applications and robots. As the core structure of the map in RTAB-Map is graph-based, it consists of nodes and links. The sensor data, and additional information like visual words and local occupancy grids is stored in the node that is created by a Short-Term Memory module. This data assists in subsequent processes such as Loop Closure and Proximity Detection, and Global Map Assembling. Nodes are generated at a fixed rate set by the parameter Rtabmap/DetectionRate, which can be adjusted based on the robot's speed and sensor range to ensure appropriate overlap between consecutive nodes as well as optimizing memory and processing requirements.

The system differentiates between three types of links: Neighbor, Loop Closure, and Proximity links, which connect the nodes in the graph. The Neighbor links are included in the Short-Term Memory module between consecutive nodes which includes the odometry transformation, while Loop Closure and Proximity links are identified through their respective detection mechanisms, which is either loop closure detection or proximity detection, which in turn provides constraints for the graph optimization. Whenever one of these two links are added, a graph optimization process is triggered, which corrects the entire graph to minimize the odometry drift. The optimized graph then has the possibility to publish OctoMap, Point Cloud, and 2D Occupancy Grid outputs to external modules.

A key advantage of RTAB-Map is the graph-based nature which allows for efficient memory usage and robust map management. This structure supports large-scale and long-term SLAM operations by dynamically managing memory and optimizing the map in real-time [4].

However, due to its flexibility and the range of features it offers, RTAB-Map can be complex to configure and tune effectively. If e.g. the detection rate parameter is set too high, it would cause an increasingly high memory usage and computation time.

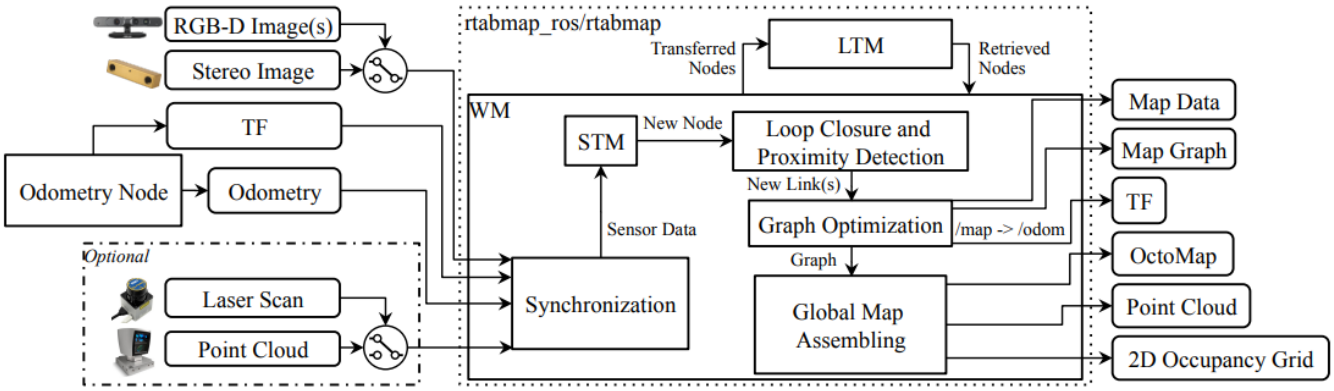


Fig. 7: RTAB-Map framework structure [4]

2.2.4. RESLAM

RESLAM [5] is a novel, edge-based SLAM system. The novelty and main advantages of RESLAM comes from its edge-based approach where it improves edge depth, optimizes the camera's poses and intrinsic parameters, while simultaneously performing loop closure on a global map as well as supporting relocalization.

The architectural framework of RESLAM is organized into four main parts that operate in parallel: System, Visual Odometry, Local Mapper, and Global Mapper. The framework structure is shown in Fig. 8. It starts by processing the RGB-D data to identify the edges. Next, the Visual Odometry part calculates how the camera moves relative to a specific frame. This information is then sent to the Local Mapper to decide whether to mark this frame as a keyframe, which if marked, is included in a local set for further analysis. This process involves refining the edges' depth and the camera's pose and intrinsic parameters. Finally, the Global Mapper looks for potential loops in the data. If a loop is detected, it attempts to close it, if not, the frame is stored for future reference. In situations where the system is put to a halt or if it encounters an error in estimating the camera's pose, it switches to relocalization mode where it tries to re-establish its location using new RGB-D data until it is successful in doing so.

There is a drawback for this system however, and this is mostly in cases where there is a lack of texture for edge detection or the depth information is missing for large parts of the scene. These cases are typically represented by excess sunlight or partial or full covering of the sensor. If any of these situations above occur, tracking losses will become a problem [5]. And despite the fact that it has relocalization capabilites when tracking losses occur, it is not a given that it will be able to relocalize after this occurs.

2.2.5. ManhattanSLAM

ManhattanSLAM [6] proposes a SLAM system that utilizes a less stringent version of the Manhattan World assumption [31] to make it applicable to both Manhattan and non-Manhattan environments through

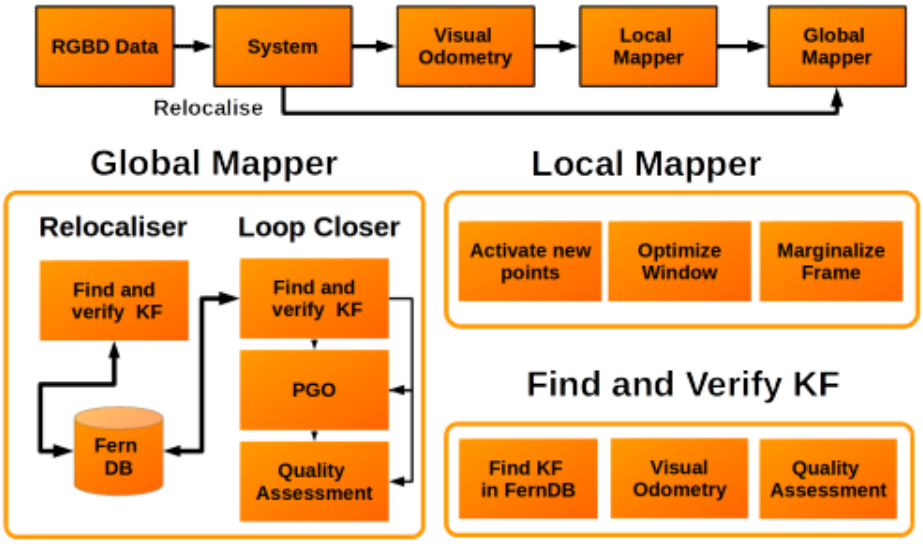


Fig. 8: RESLAM framework structure [5]

a combination of point, line, and plane tracking. It offers a method to detect Manhattan Frames from planes to model a scene as a Mixture of Manhattan Frames. Manhattan Frames are coordinate systems that are aligned with the primary orthogonal axes of the environment, such as the edges of buildings and streets, which form a grid-like pattern. In turn, it results in drift-free rotation estimation, alongside robust feature tracking in situations where Manhattan Frames are not present. This serves as one of the main advantages of ManhattanSLAM, since maintaining high accuracy in the rotation estimation without the accumulation of errors is very crucial in SLAM applications.

Additionally, it introduces an efficient surfel-based mapping strategy that optimizes memory use by differentiating between planar and non-planar regions. This allows for a more memory-efficient representation of the environment, as planar regions can be densely packed with fewer surfels without loss of detail, reducing the computational load and memory usage [6].

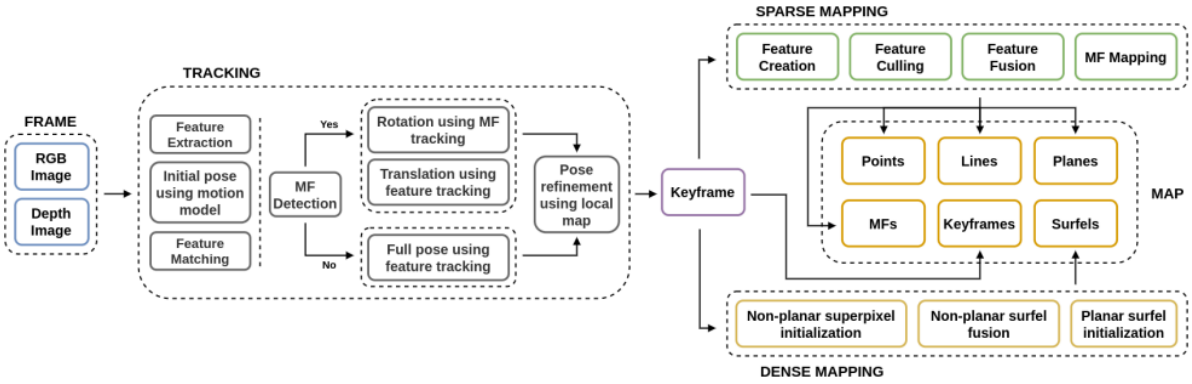


Fig. 9: ManhattanSLAM framework structure [6]

2.3. LIDAR-based SLAM approaches

There are a plethora of different approaches based on LIDAR and as such this thesis will only explore a handful of the alternatives. A more comprehensive list of the approaches can be found at github [32]. For each SLAM system, the basics of how it works will be introduced, as well as some upsides and downsides to each.

2.3.1. A-LOAM, Advanced LIDAR Odometry And Mapping

Advanced LIDAR Odometry and Mapping (A-LOAM) is based on the original LOAM [33]. A-LOAM serves as the benchmark and to show how the other algorithms iterate and differentiate from the original implementation.

Each scan from the LIDAR is processed by two nodes in parallel. The LIDAR odometry estimates the motion of the LIDAR between two consecutive steps. While these estimated steps are used to correct for distortion, and then the non-distorted scans are combined to create a global map [33].

The benefits of A-LOAM is considered similar to the the original LOAM with improved performance due to the implementations of optimization libraries such as Eigen [34] and Ceres Solver [35]. A-LOAM uses feature extraction from sharp edges and planar surfaces to optimize the matching process. This reduces the number of points which needs to be compared while minimizing the loss of information. This feature extraction can lead to reduced performance if the feature extraction removes too many points. It does not offer any loop closure, meaning that there is no compensation for drifting in the map.

While the original implementation of LOAM is closed source, A-LOAM [36] is open-source and is the basis for some of the other implementations that will be introduced, such as SC-A-LOAM, LeGO-LOAM and F-LOAM.

Fig. 10: Resulting map after running A-LOAM on KITTI sequence 00. Image from their GitHub [36].

2.3.2. *F-LOAM, Fast LIDAR Odometry And Mapping*

The authors and developers of F-LOAM wanted to improve the original LOAM by reducing the computational cost. This is achieved by approaching distortion compensation without using iterative computation. Instead they assume that the angular and linear velocities are constant in between the scans thus simplifying the computation. They also increase the efficiency of the feature extraction and matching by utilizing a local map and adding a weighted optimization for pose estimation. By not feature matching on a global map they are able to reduce the computations even further which allows the algorithm to run faster. In addition, weighed optimization allows the method to use the most significant local geometries to be prioritized in the matching process. [37]

These improvements are even more significant when there is an increase in either duration or length traveled. Such improvements are especially good for real time applications as there is often limited hardware resources on robots. Assumptions about constant angular and linear velocities could be exposed as weakness when dealing with data containing more inconsistent movement patterns.

2.3.3. *SC-A-LOAM, Scan Context Advanced LIDAR Odometry And Mapping*

As there are no papers regarding this specific algorithm, a more detailed description of how scan context works and how it improves upon A-LOAM will be described. This extension of A-LOAM aims to add place recognition abilities and add loop detection and loop closure.

Scan Context++ introduces new techniques to introduce the place recognition capabilities for A-LOAM (section 2.3.1). It aims to tackle how changes in lateral movements and rotations often leave the algorithms struggling when recognizing places by introducing augmented descriptors. These descriptors are used to compare the current position to previously visited positions given a distance function and similarity constraints. To help the algorithms with estimating pose within a previously visited location, it provides an initial guess for the metric pose. The framework can be seen in Fig. 11 where the semi-metric Localization represents the initial guess of the method [38].

This addition of scan context on top of A-LOAM allows for submap handling, which in turn allows the SC-A-LOAM to not update the global map as frequently, and thus reducing computational cost. Further enhancements are the ability to optimize these submaps independently.

The increased complexity increases the difficulty of the parameter tuning as there are more parameters. While the integration and merging of submaps can introduce errors if the transformation between the submaps is not accurate.

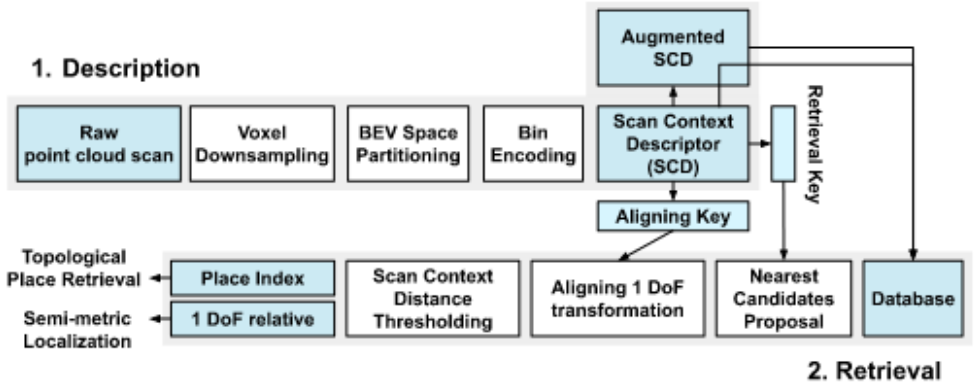


Fig. 11: Describes how the framework for SC-A-LOAM processes data from the pointcloud. The semi-metric localization is the estimated position within the map. [38]

2.3.4. LeGO-LOAM, lightweight and ground-optimized

Lightweight and Ground-optimized LIDAR Odometry and Mapping (LeGO-LOAM) is a SLAM system that outputs an estimated pose with 6 degrees of freedom. It projects the input scan into a range image and extracts features from this image. The feature extraction also removes noise from the scan and the package has parameters that can be tuned in environments where the scans might be more noisy. An example of noise reduction can be seen in Fig. 12. These features are also used to calculate the trajectory between the consecutive scans. LeGO-LOAM also maps these features to a global cloud map. This method is specifically developed for use with ground based systems [39].

This ground optimization is supposed to increase performance where there is height variance within the terrain [39]. One potential drawback is an overly aggressive pre-filtering, as this might remove points which support scan-matching.

2.3.5. HDL Graph SLAM

HDL Graph SLAM [40] is an open-source package for SLAM. As the name implies it relies on graphs to represent the environment as the robot moves about. It uses poses over time, combined with scans to keep track of landmarks [41]. This method also applies NDT [42] scan matching-based odometry to optimize its scan matching capabilities.

This package is developed for indoor and outdoor applications and consists of four nodes. The flow between these nodes can be seen in Fig. 13. The odometry estimation that is used to estimate the trajectory is within the scan_matching_odometry node. An example of a global map generated by HDL graph SLAM can be seen in Fig.14 where the color of the ponts shows ther height[40].

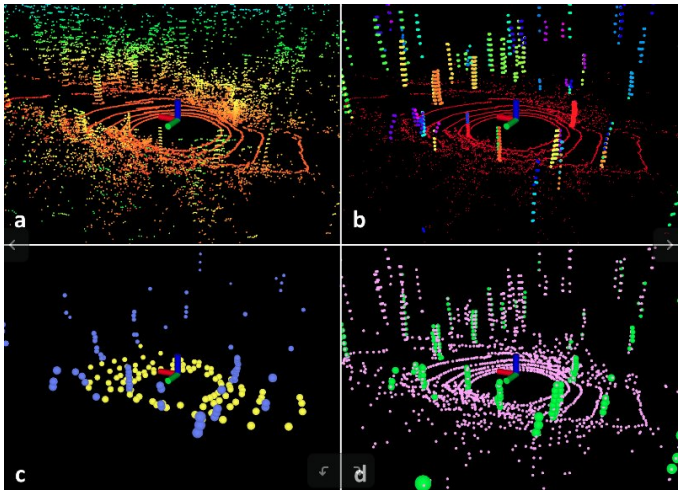


Fig. 12: Example of how LeGO-LOAM handles a messy scan using feature segmentation. The original scan is shown in (a). (b) the red points are the ground while the rest of the points are the points remaining after the segmentation. (c) the blue and yellow dots represents edges and planar features. Lastly, (d) represent the green and pink dots represent the edges and planar features in global frame. [39].

One significant upside is the efficient loop-closure detection which allows the map to update and compensate for drifting in the map generation. A downside with HDL Graph SLAM is the computational cost to process and optimize the graph. In real-time applications this graph optimization can lead to significant delays between scans as it tries to optimize the graph.

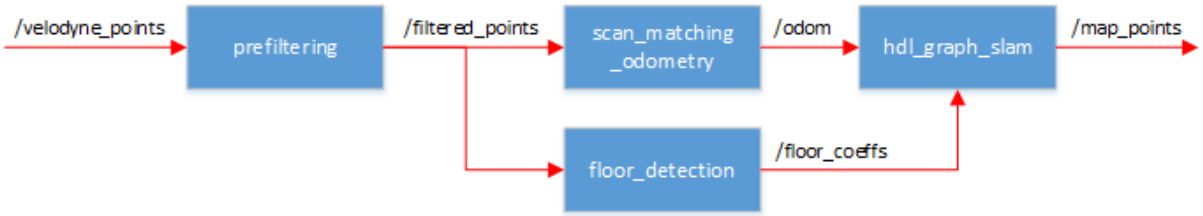


Fig. 13: HDL Graph SLAM data flow through the nodes [40].

2.3.6. KISS-ICP

Keep It Small and Simple Iterative Closest Point, or just KISS-ICP [43], provides an alternative based on ICP [20] to estimate a trajectory based only on 3D LIDAR scans. For each frame, it aims to compute the LIDAR trajectory by comparing consecutive scans. To estimate the trajectory without any other sensors, it relies on a constant velocity model which assumes that the sensor moves at a constant velocity between the scans.

One upside of KISS-ICP is that it does not require parameter tuning between different environments and platforms as shown in Fig. 15. KISS-ICP also adds algorithms to predict motion to aid in the deskewing

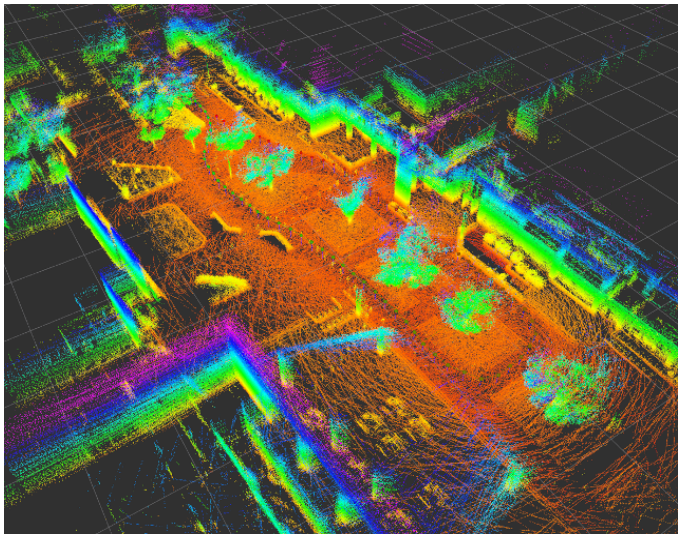


Fig. 14: Example of global map as created by the example rosbag (hdl_400.bag) provided by the publishers on GitHub [40].

and movement compensation.

A drawback of KISS-ICP is that it does not offer any loop closure mechanism, limiting how it can deal with cumulative errors over time. It is also not developed for SLAM, only for odometry estimation.

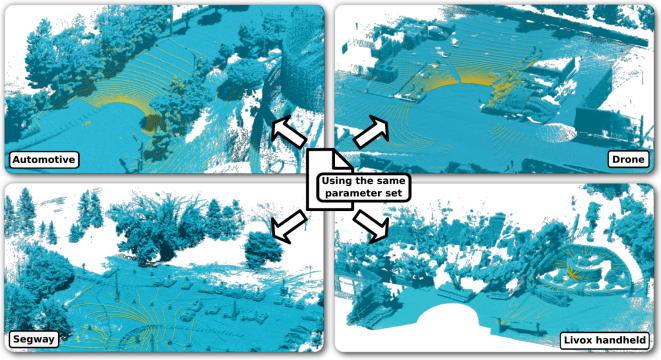


Fig. 15: Illustration of the mapping performed by KISS-ICP without tuning the parameters for each setting. Courtesy of [43].

2.4. Sensing technologies

2.4.1. Depth Camera

Stereo depth cameras could be compared to human vision in the sense that it uses two cameras placed a few centimeters apart. They spot the same points on each camera and then calculates how far away these points are by measuring the differences between their positions, which is a method called triangulation. When the camera comes across a featureless surface, such as a flat, white wall, it may sometimes be

an issue, but more often than not, the camera is equipped with a light projector that shines a patterned infrared light to aid in finding the points [44]. Many stereo depth cameras also have the ability to capture RGB information, allowing them to add color to the depth data they collect, such as the Intel RealSense cameras. And as such, in this thesis, a Intel RealSense D435i is used to collect data. A picture of the camera is shown in Fig. 16.

Fig. 16: Image of Intel RealSense D435i [7]

2.4.2. 3D LIDAR

Light Detection and Ranging (LIDAR) is an active sensor, by emitting light waves and calculating the time between emitting and detection one can get the position of objects within the environment [45]. There are several types of LIDAR's but this thesis utilizes a surround LIDAR. Surround LIDAR's are generally a scanner that rotates with a high frequency and combines the scans as it rotates to create a 3D representation of the environment.

The LIDAR that was utilized for this thesis is an Ouster OS0-128 3D LIDAR as seen in Fig. 17.

2.5. ROS (Robot Operating System)

ROS [47] is an open-source framework with various tools and libraries to write robot software and develop robot applications. Researchers from Stanford University and University of Southern California developed a standard for robotics development to make development more consistent and transparent. The goal of ROS was to develop a free, open-source and tool-based framework which can support different coding languages.

2.5.1. Data representation, PointCloud2

To represent 3D range data with ROS the default sensor message is given as shown in Fig. 18.

Where the header contains when the data is send and which frame_id it is related to. Height and width describes the size of rows and the number of rows each row contains. Fields are a array of pointfields

Fig. 17: Image of Ouster OS0-128, a rotating 3D LIDAR with 128 vertical channels [46].

```
std_msgs/Header header
uint32 height
uint32 width
sensor_msgs/PointField[] fields
bool is_bigendian
uint32 point_step
uint32 row_step
uint8[] data
bool is_dense
```

Fig. 18: Example of a PointCloud2 message in ros [8].

where each pointfield describes one dimension (such as x, y, z, or intensity). Data represents the actual data and the rest of the fields describe how the data passed and represented. [48].

2.5.2. Rosbag

Rosbag [47] is a tool that is used for recording and playback of data from ROS topics. It also offers many different functionalities like compressing data to save space for big recordings, as well as making it possible to filter out what topics should/should not be recorded/played. Additionally, it is also possible to display information about the rosbag that is human-readable using the command ”rosbag info” [49]. This allows for easy confimation that the data has been recorded as expected. An example of how this would look like is shown in Fig. 19.

```
$ rosbag info foo.bag path: foo.bag
version: 2.0
duration: 1.2s
start: Jun 17 2010 14:24:58.83 (1276809898.83)
end: Jun 17 2010 14:25:00.01 (12768099900.01)
size: 14.2 KB
messages: 119
compression: none [1/1 chunks]
types: geometry_msgs/Point [4a842b65f413084dc2b10fb484ea7f17]
topics: /points 119 msgs @ 100.0 Hz : geometry_msgs/Point
```

Fig. 19: Example of the output for ”rosbag info”.

2.5.3. Rviz

Rviz is a 3D visualization tool that is used to display various topics that are published from a node, these include information from sensor data, robot models, and more. It is very versatile and there are a multitude of parameters to configure which can be saved as a .rviz config file for later use. Furthermore, rviz supports a variety of data types such as point clouds, images and laser scans, making it useful for different applications [50].

2.6. RTK-GNSS

Real-time Kinematic Positioning (RTK) is a high precision technique used to increase the accuracy of Global Navigation Satellite System (GNSS). Compared to standard GNSS which can reach meter-level accuracy, a RTK system can reach centimeter-level accuracy. This accuracy makes it suitable for creating ground truth trajectories when collecting data outside.

To increase accuracy, a RTK system relies on a base station, where a stationary receiver is located at a precisely known coordinate. This base station is used to calculate correction data based on the received satellite signals with the base station. The mobile robot also needs a receiver that can receive correction data from the base station [51].

While RTK is capable of high accuracy position estimates, it can only do so if it receives a clear signal from both the base station and satellites. The system provides the covariance matrix for error estimation for longitude, latitude and altitude. Within ROS, these error estimates can be given in the format as shown in Fig. 20 as the variable position_covariance.

The estimated pose error is given in a 3x3 matrix with the following format:

Message Type: nav_msgs/NavSatFix
Fields:
`Header header` (seq, stamp, frame_id)
`NavSatStatus status` (status, service)
`float64 latitude, longitude, altitude`
`float64[9] position_covariance`
`uint8 position_covariance_type`

Fig. 20: Compact ROS Message Type Specification for nav_msgs/NavSatFix [9].

$$\mathrm{Cov}(X) = \begin{bmatrix} \sigma^2_{\text{lat}} & \sigma_{\text{lat,lon}} & \sigma_{\text{lat,alt}} \\ \sigma_{\text{lat,lon}} & \sigma^2_{\text{lon}} & \sigma_{\text{lon,alt}} \\ \sigma_{\text{lat,alt}} & \sigma_{\text{lon,alt}} & \sigma^2_{\text{alt}} \end{bmatrix},$$

where the variance in the diagonal elements (σ^2_{lat}, σ^2_{lon}, σ^2_{alt}) is the variances for longitude, latitude and altitude squared. They represent the squared uncertainty for each of the dimensions, while the off-diagonal elements represent the covariance between the different dimensions.

2.7. Evaluation metrics

In SLAM, evaluating the performance of a system is crucial for understanding its accuracy, robustness, and reliability. One common metric used for this purpose are the Absolute Pose Error (APE). This metric provides insights into how well a SLAM system can track its position and orientation over time.

For evaluating and comparing the trajectory of the various SLAM approaches in this thesis, a Python package named evo was used. This package supports multiple data formats, which includes KITTI, TUM and EuRoC MAV, as well as support for ROS1/ROS2 rosbags [52].

2.7.1. Absolute Pose Error

The Absolute Pose Error measures the absolute difference in the pose between the estimated trajectory provided by the SLAM system and the ground truth trajectory. This metric is useful for understanding the overall accuracy of the system across a complete dataset. To calculate the APE, you first have to align the estimated trajectory with the ground truth trajectory. This alignment can be done using methods such as the Umeyama method [53], which is used in the evo package [52]. These alignment methods provide a transformation T that best aligns the estimated trajectory with the ground truth. This transformation includes rotation translation and in the case of Umeyama, possibly a scale factor. Let's denote the transformation resulting from the Umeyama method as T_{align}, which applies to the estimated poses to align them with the ground truth before computing the APE. If we consider N to be the total number of timestamps at which

poses are estimated and ground truth is available, the RMSE for APE, after alignment, across all time indices can be expressed as:

$$RMSE_{APE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\|q_{gt}(i) - T_{align} \cdot q_{est}(i)\|\right)^2}, \tag{4}$$

where $q_{gt}(i)$ and $q_{est}(i)$ is the ground truth pose and estimated pose at timestamp i. [54]

2.7.2. Map density

To calculate the map density, this thesis utilizes a volume-based density calculation method. This is done by finding the volume of the boundary box for the point cloud and divide the number of points in the point cloud by this volume. This assumes that all the resulting maps have the same scale.

To achieve consistent method of calculation across all systems and data sequences, the following method was applied:

$$\frac{N}{V}, \tag{5}$$

where N is the total number of points and V is the volume of the bounding box.

3. METHODOLOGY

This section presents the hardware and software used, which includes the robotics platform for data collection and the computers that ran the SLAM systems, as well as what and how each software was utilized. Subsequently, the benchmark datasets are introduced, featuring TUM-RGBD for camera and KITTI for LIDAR. Following this, the data that was collected is described, with separate sections on the orchard environment and the parking area. The process of generating ground truth for the collected dataset is also discussed. Moving along, the integration of both camera-based and LIDAR-based SLAM systems are then further discussed w.r.t how they were utilized with all of the datasets. Lastly, this chapter introduces how the calculations of the density and volume of the generated maps were calculated. **All of the scripts and configuration files described in this chapter are provided in the Appendix.**

3.1. Hardware

The hardware section will be split into different parts; first we will introduce our robotics platform which was used to record the orchard and parking dataset. Then we will separate the hardware used to run camera based and LIDAR based algorithms as these were ran on two different computers.

3.1.1. Robot Platform

RB-VOGUI [55] is a robot developed by Robotnik and this was the one used when collecting the data. The specifications of the robot are listed under:

- **Dimensions:** 1.040 x 650 x 530 mm
- **Weight:** 165 Kg
- **Max Velocity:** 2,5 m/s
- **Environment:** Indoor/Outdoor
- **Enclosure Class:** IP50
- **Autonomy:** Up to 6 h
- **Batteries:** LiFePO4 30Ah@48V
- **Traction Motors:** 4 x 500 Watts with brake
- **Steering Motors:** 4
- **Temperature Range:** -10°C to +45°C
- **Payload:** Up to 150 Kg

- **Maximum Slope:** 47
- **Controller:** Industrial PC Intel i7, Open Architecture based on ROS
- **Connectivity:** USB, RJ45, HDMI, Power supply 12, 24, VDC and batteries
- **Operating System:** Ubuntu 18.04
- **Depth Camera:** Intel Realsense D435i
- **3D-LIDAR:** Ouster OS0-128
- **GNSS:** u-blox C099-F9P-1 application board with ZED-F9P module

Although most of the components are delivered with the robot, the 3D LIDAR was manually mounted and configured by us, and in doing so, it was necessary to also configure the TF transformations and launch files to make it operable with the robot. The picture of our modified RB-Vogui robot is shown in Fig. 21.

3.1.2. Computer hardware for camera-based SLAM methods.

- **Processor (CPU):** Intel i5-12400F 6-core Processor, 12 threads, 2.5GHz/4.4GHz
- **Graphics Processing Unit (GPU):** NVIDIA GeForce RTX 3060
- **Memory (RAM):** 16 GB
- **Storage:** 1 TB SSD
- **Operating System:** Host PC: Windows, Guest: Ubuntu 16.04, 18.04 and 20.04 on Oracle VM VirtualBox

3.1.3. Computer hardware for the LIDAR-based SLAM methods.

- **Processor (CPU):** AMD Ryzen 7 3700X 8-Core Processor, 3600 MHz, 8 core(s), 16 logical processor(s)
- **Graphics Processing Unit (GPU):** NVIDIA GeForce 1660 SUPER
- **Memory (RAM):** 32 GB
- **Storage:** 2 TB HDD & 1 TB SSD
- **Operating System:** Ubuntu 18.04 running as Windows Subsystem for Linux (WSL) in Windows 10
- **Note:** Default WSL configuration allocates a maximum of half of the total RAM capacity and hence it effectively only had 16 GB RAM while running the algorithms.

Fig. 21: RB-Vogui with attached Ouster OS0-128 on top and RealSense D435i integrated in the front.

3.2. Software

This section aims to introduce the different software that was applied throughout this thesis. Some of which help to create a stable environment for code execution, while others help with data visualization and debugging.

3.2.1. Oracle VM VirtualBox

Oracle VM VirtualBox [56] was used to run different versions of Ubuntu such that they would be compatible with all of the camera-based SLAM systems. It was allocated 10GB of RAM to each Virtual Machine (VM) with 100GB of storage for each. The VMs were also setup with 4 virtual CPUs, corresponding to 4 threads, to ensure efficient utilization of the host's Intel i5-12400F CPU, which features 6 cores and 12 threads.

3.2.2. Docker

Docker [57] was used to containerize the evaluation environment for LIDAR based methods. Docker allows the program to run in an isolated environment from the host computer while dynamically being allocated the amount of computing resources depending on the requirements of the container. This creates a stable environment that can also be quickly implemented on other systems, increasing the reproducibility.

3.2.3. Cloud Compare

For our research, we used CloudCompare [58], which is an open-source software for 3D point cloud processing, to inspect the full point cloud and allow a closer inspection of areas with worse results. This software allowed us to catch errors in configurations from mapping attempts, and thus aided our research to get more accurate results.

3.2.4. ChatGPT

For the implementations of the algorithms, ChatGPT was used to help debug errors during setup of the SLAM systems and to help ensure that the configuration files matched our use case. ChatGPT was also used to speed up the development of certain scripts that were used for various purposes. For correcting and refining any grammatical errors, ChatGPT was also used.

3.3. Benchmark Experiments

As there are no available datasets that we are aware of which have both RGB-D and LIDAR data, we used two different datasets to establish benchmark performance. Four sequences from TUM and two sequences from KITTI. This section will introduce each of them and explain what the sequences contain.

3.3.1. TUM-RGBD benchmark experiment

To obtain a baseline of how each of the SLAM systems perform, it was first implemented and evaluated on the TUM RGB-D dataset [14]. This dataset includes RGB-D data that was recorded with a Microsoft Kinect, alongside a ground-truth trajectory that was obtained from a motion-capture system. There are multiple sequences that contain static and dynamic scenes, both with and without loop closure. The aim was to evaluate how each SLAM system performed in various scenes, therefore it was tested on four different sequences in this thesis. These include the fr2_desk, fr2_desk_with_person, fr3_walking_xyz and fr3_sitting_static sequences. A more descriptive explanation for each of the sequences is provided below.

fr2_desk:

- Captures a traditional office environment containing two desks with standard office items.
- The Kinect device navigates around the tables, completing a full circuit such that the loop is closed, thus being useful for investigating loop closure capabilities of SLAM systems.

fr2_desk_with_person:

- Similar to the *fr2_desk* setup, but includes an individual seated at one of the desks.
- The person interacts with various objects during the recording, adding dynamic elements to the scene.

fr3_walking_xyz:

- Includes two individuals moving in the scene while simultaneously capturing movement along the x, y and z axes.
- Focuses on evaluating dynamic objects that are moving quickly in the scene.

fr3_sitting_static:

- Contains a static scenario with two individuals sitting and talking.
- Focuses on evaluating dynamic objects that move slowly in the scene.

3.3.2. KITTI benchmark experiment

To compare how well the LIDAR based SLAM methods work, they were first tested on two sequences from the KITTI dataset. KITTI is one of the most popular datasets for developing and testing SLAM methods [13]. As there are some different versions of the KITTI dataset we went with the odometry dataset as this contains the ground-truth poses that were obtained with a GPS/OXTS sensor. As all methods for LIDAR are applied using Robot Operating System (ROS) we converted two sequences from the KITTI dataset into

rosbags. That is, the first and fifth sequence. Both of these rosbags contain the same number and type of rostopics, but the environments in which they drive are quite different.

The first sequence starts at a T-intersection and drives down a high way. There are no loops in this data and hence we can not utilize loop detection. The average velocity for this sequence is 78.2523 km/h and the distance traveled is 2469 meters.

The fifth sequence is from a more populated area and this contains multiple places in which the car returns to the same street which allows for loops to be detected. The average velocity for this sequence is 27.5858 km/h and the distance traveled is 2200 meters.

A more comprehensive comparison between the sequences can be found in table 1. While the distance traveled is similar in both of these sequences the speed, duration and the environment are different.

	Sequence 01	**Sequence 05**
Duration (s)	113.6137	287.1200
Path Length (m)	2469.4728	2199.9117
Statistics		
Average Time Interval (s)	0.1037	0.1042
Maximum Time Interval (s)	0.1063	0.1103
Minimum Time Interval (s)	0.1011	0.0982
Average Velocity (km/h)	78.2523	27.5858
Average Velocity (m/s)	21.7367	7.6627
Maximum Velocity (km/h)	112.5752	42.3820
Maximum Velocity (m/s)	31.2709	11.7728
Minimum Velocity (km/h)	31.1331	0.0262
Minimum Velocity (m/s)	8.6481	0.0073

Table 1: Comparison of Sequence Statistics for the KITTI[13] dataset. This is obtained by utilizing the EVO[52] python package and comparing the ground truth in both sequences.

These two sequences allow us to demonstrate if the methods work on our computer and establish a reference performance.

3.4. Collected data

For this thesis, data was collected from two different settings: an orchard, representing an agricultural environment, and a parking lot, representing an urban environment. Both of these drove a route where the robot ends approximately where it started. Hence, both had the capability of creating closed loops. The complete information can be found in Table 2, where the values were generated using the evo package [52]. The locations of the recording from a satellite view can be inspected in Fig. 22.

Fig. 22: Map of the locations of data collection, with pins at the locations. Image created using Google maps [59].

	Orchard	**Parking**
Duration (s)	357.692	164.202
Path Length (m)	156.651	151.358
Statistics		
Avg Time Interval (s)	0.100	0.1001
Max Time Interval (s)	0.109	0.110
Min Time Interval (s)	0.092	0.089
Avg Velocity (km/h)	1.577	3.320
Avg Velocity (m/s)	0.438	0.922
Max Velocity (km/h)	2.575	5.048
Max Velocity (m/s)	0.715	1.402
Min Velocity (km/h)	0.000	0.036
Min Velocity (m/s)	0.000	0.010

Table 2: Trajectory statistics for the orchard and parking lot environments using the evo python package.

3.4.1. Orchard

This dataset was recorded in an orchard on the campus of NMBU, and it totaled about 6 minutes. The weather was shifting between sunny and cloudy during the recording, and the terrain was very uneven and slightly muddy due to the weather being rainy a couple of hours prior to recording. The path length is 156.6 meters and the average velocity is 1.577 km/h. In regards of altitude, the first half of the trajectory is downhill and the second half is driving back up to the original altitude.

3.4.2. Parking lot

This recording was captured outside the robotics lab on the NMBU campus. The terrain is paved, but because the pavement is a bit worn, it is still slightly bumpy at times. The weather at the time of the recording was dry and sunny. The path length is 151.4 m and the average speed is 3.3 km/h. The altitude is close to flat, apart from small bumps on the road.

3.5. Ground truth generation for collected datasets

Due to the different data formats between the camera based approaches and LIDAR based approaches the global position to local coordinates was handled differently.

3.5.1. Camera-based SLAM systems

As we are utilizing a RTK-GNSS to create a ground truth, the recorded data from the GPS will be stored as latitude, longitude and altitude. Moreover, when the SLAM systems are predicting a trajectory estimate, they are predicting poses (x, y, z, qx, qy, qz, qw), meaning that if someone would evaluate the predicted poses with the GPS coordinates, it would not make sense. In order to counteract this, the solution is to convert the GPS coordinates into Cartesian coordinates such that the ground truth is in the same format as the estimated trajectory. There were multiple steps involved in this process, the first one was to extract only the */robot/gps/fix* topic into a .txt file by firstly using this command in a Bash terminal after running *roscore* in another terminal first:

```
rostopic echo foo.bag > gps_extracted_from_bag.txt
```

Then, you would playback the rosbag like this:

```
rosbag play --clock foo.bag
```

This is done such that it can write to the .txt file. When this is done, a script was created in order to read the GPS coordinates for every timestamp and output it in a format like this:

```
timestamp longitude latitude altitude
```

Fig. 23: Image of orchard environment where data was collected. The robot was driven down the row between the trees before making a U-turn at the bottom to drive back up on the right-hand side of the picture.

Fig. 24: Image of the parking lot where data was collected. The recording was performed a few days later when there were more parked cars.

Then another script was created in order to convert the GPS coordinates into cartesian coordinates (x, y, z). In addition to this, since the camera is fixed in the collected dataset, the constant orientation means there is no variation in rotation. Thus, the script also adds 0 0 0 1 to the end of every line such that the quaternions represent a neutral rotation in 3D space. When all of this is done, a ground truth for the recorded bag file has been generated.

3.5.2. LIDAR-based SLAM systems

For the LIDAR based system, one also had to convert to Cartesian local coordinates. To achive this the GPS coordinates from the rostopic */robot/gps/fix* was used to generate a new topic in the rosbag containing the local coordinates. To convert from global cordinates to local cordinates the geographiclib [60] python package was used. When converting from global to local coordinates has to define where the origin of the coordinates are. To define this, the position of the first gps message in the rosbag was defined as the origin. As we only had a singular RTK-GNSS reciever we can only consider the position, not the orientation.

3.6. Setup of SLAM systems

As all systems needed alteration to allow the algorithms to work on both benchmark datasets and the collected dataset, this section aims to explain what changes were made to each system.

3.6.1. Setup of camera-based SLAM systems

All of the camera-based SLAM systems were built and compiled with the instructions found on their respective GitHub pages ([61], [62], [63], [64], [65]). For ORB-SLAM2, ManhattanSLAM and RESLAM, the only adjustment that was made in the .yaml configuration files, was the camera's intrinsic parameters. Since RTAB-Map and RGBDSLAMv2 were configured with ROS, the only modifications that were done in the launch files was the adjustment of rostopics as well as the camera's intrinsic parameters. **Note; these adjustments that were mentioned is only applicable for the collected dataset.**

3.6.1.1. Conversion of collected rosbag to TUM format

Since ORB-SLAM2, ManhattanSLAM and RESLAM, in contrast to RGBDSLAMv2 and RTAB-Map, were not ran using a rosbag, there had to be a conversion of the rosbag to a proper format, more specifically the TUM format. This includes a RGB and depth folder as well as an associations file that contains the synchronized timestamps between each RGB and depth image. An example of how one line of the associations file looks like is given here:

```
1712929696.511281 rgb/1712929696.511281.png 1712929696.511281
↪  depth/1712929696.511281.png
```

The recorded topics of the rosbag that had to be converted was */robot/camera/color/camera_info*, */robot/-camera/color/image_raw* and */robot/camera/depth/image_rect_raw*. These had to be converted into a RGB and depth folder containing their respective images using a script . After this was done, a *rgb.txt* and *depth.txt* file had to be made using this script such that it could be used with the association script provided by TUM, found on this page [66]. This association script generated an *associations.txt* file as exemplified above and the collected dataset was almost ready to be used with ORB-SLAM2, ManhattanSLAM and RESLAM.

Note; **All** of the following commands in this section (until the section about TUM-RGBD benchmark experiment) are to be executed within a terminal using Bash. For the commands that utilize roslaunch, it is important to source the workspace as well before executing the command.

3.6.1.2. ORB-SLAM2

For running and evaluating the sequence on the TUM-RGBD dataset, the configuration file(s) named TUMX.yaml is included in the repository, meaning the SLAM system is ready for use if you have installed the dataset, with this command:

```
./Examples/RGB-D/rgbd_tum Vocabulary/ORBvoc.txt Examples/RGB-D/TUMX.yaml
↪  PATH_TO_SEQUENCE_FOLDER ASSOCIATIONS_FILE
```

For running and evaluating the sequence on the collected dataset, the command is pretty much similar except that the configuration file changes from TUMX.yaml to D435i.yaml, meaning that the command would be:

```
./Examples/RGB-D/rgbd_tum Vocabulary/ORBvoc.txt Examples/RGB-D/D435i.yaml
↪  PATH_TO_SEQUENCE_FOLDER ASSOCIATIONS_FILE
```

One thing to note; the ORB-SLAM2 package that was used in this thesis is a modified version of the original one. The only difference is that it includes a reconstructed point cloud map, whereas the original does not. However, this does **not** impact the estimated trajectory or any other aspects than the reconstructed map.

3.6.1.3. ManhattanSLAM

ManhattanSLAM is based upon ORB-SLAM2, meaning that the command to run the SLAM system is almost identical to ORB-SLAM2. This also entails that the same process for converting the rosbag to TUM format for the collected dataset had to be done here. For running ManhattanSLAM on the TUM-RGBD dataset, the command would like this:

```
./Example/manhattan_slam Vocabulary/ORBvoc.txt Example/TUMX.yaml
↪  PATH_TO_SEQUENCE_FOLDER ASSOCIATIONS_FILE
```

For running it on the collected dataset, the command would like this:

```
./Example/manhattan_slam Vocabulary/ORBvoc.txt Example/D435i.yaml
↪  PATH_TO_SEQUENCE_FOLDER ASSOCIATIONS_FILE
```

3.6.1.4. RESLAM

While RESLAM, in addition to ORB-SLAM2 and ManhattanSLAM, had to have an *associations.txt* file to run the SLAM system, this was not entirely equal in terms of this. The command to run the SLAM system on the TUM dataset looks like this:

```
build/RESLAM config_files/reslam_settings.yaml config_files/dataset_tumX.yaml
```

Whilst running it on the collected dataset, the command would look like this:

```
build/RESLAM config_files/reslam_settings.yaml config_files/D435i.yaml
```

3.6.1.5. RTAB-Map

In RTAB-Map we see a different approach than in the three, aforementioned SLAM systems, since rosbags are utilized here instead of using an *associations.txt* file. Firstly, a launch file had to be created to be

compatible with the collected dataset. Then, to run the SLAM system, one would execute this launch file, then proceed to Rviz and add the topics of interest for visualization. Finally, run the rosbag using this command:

```
rosbag play --clock foo.bag
```

After the rosbag has finished processing, the reconstructed map and estimated trajectory is stored in RTAB-Map's database, and this can be accessed using this command:

```
rtabmap-databaseViewer ~/.ros/rtabmap.db
```

This opens a GUI to view the database of the processed rosbag, and the main points of interest here is exporting the estimated trajectory as well as the reconstructed map. This is easily done using "Export poses" and "Export 3D map" under "File" inside the GUI.

3.6.1.6. RGBDSLAMv2

Similar to RTAB-Map, this SLAM system also operates with rosbags, meaning that a launch file had to be created here as well to make sure it was able to be ran with the collected dataset. For running it with the TUM-RGBD dataset, a launch file was provided with the repository and the command would look like this:

```
roslaunch rgbdslam test_settings.launch
↪  bagfile_name:="/home/marko/Documents/rgbd_dataset_freiburg2_desk.bag"
```

To run RGBDSLAMv2 with the collected dataset, the command would look like this:

```
roslaunch rgbdslam realsense_slam.launch
↪  bagfile_name:="/home/marko/Documents/orchard.bag"
```

When the bag file has finished processing, the GUI in which the SLAM system was visualized, there are options to export the trajectory and reconstructed map, similar to RTAB-Map. Here you would use "Save as Point Cloud ..." and "Save Trajectory Estimate..." under the "Save" option in the GUI.

3.6.2. Setup for LIDAR based SLAM system

While our goal was to run all the methods as they are available on GitHub, there were still some changes made to the codebases and the following section aims to clarify what changes were made to each LIDAR based approach. Although most of the packages needed to be updated to allow for input with 128 channels, some required further alteration. To install all packages, the instructions are available on each respective GitHub ([40], [36], [67], [68], [40], [43]) then the following modifications were made to each of them.

Important to note that the rosbags were played back in real time, and all were ran using roslaunch.

3.6.2.1. A-LOAM

A-LOAM did not support more than 64 vertical input channels by default, so `scanRegistration.cpp` had to be modified to allow for 128 channel LIDAR input and change the topic name to be a ROS parameter instead of being hard-coded to work only with KITTI dataset. The global map this method produced had some limitations in regard to the maximum number of points. By updating the `laserCloudNum` variable in `laserMapping.cpp` the maximum size of the resulting map was increased.

3.6.2.2. F-LOAM

The unit did not support the 128 channel LIDAR input, so changed the code within `laserProcessingClass.cpp` to allow for more channels. To allow the input topic to be changeable, the hard-coded topic name was changed for a ROS parameter that allowed alternative input topic name to be changed from launch files.

3.6.2.3. SC-A-LOAM

As SC-A-LOAM is based on A-LOAM, all the same changes had to be made to this package. Update to a ROS parameter for the input topic name and update to allow 128 channel LIDAR input, both of these modifications were made to the `scanRegistration.cpp`. This also lacked the ability to create global maps for all datasets without increasing the `laserCloudNum` variable within `laserMapping.cpp`.

3.6.2.4. LeGO-LOAM

To run this on the KITTI dataset the only file that needed to be updated was to change the `utility.h` to specify the type of LIDAR. The settings for Velodyne HDL64 was included in the file. For the ouster LIDAR used in the collected dataset the following modifications had were implemented. In the `utilily.h` the input topic was changed. The parameters for OS0-128 were updated according to Listing 1. And in `imageProjection.cpp` line 166 was uncommented so that the resulting point cloud would get the correct timestamp [69].

Listing 1: Snapshot of the utilily.h header file defining LIDAR parameters for OS0-128

```
// Ouster OS0-128
extern const string LIDAR_TYPE = "Ouster OS0-128";
extern const int N_SCAN = 128;
extern const int Horizon_SCAN = 1024; //2048
extern const float ang_res_x = 360.0 / float(Horizon_SCAN);
```

```
extern const float ang_res_y = 90.0 / float(N_SCAN-1); //45
extern const float ang_bottom = 22.5 + 0.1; //22.5
extern const int groundScanInd = 30;
```

3.6.2.5. HDL Graph SLAM

The only modifications necessary in HDL Graph SLAM was within `hdl_graph_slam_kitti.launch`. For the KITTI dataset, the static transform publisher was changed to go between vehicle and velodyne. And within the prefiltering_nodlet the base_link_name was updated to vehicle. For the collected dataset, the static transform publisher was not necessary and the base_link_frame in the prefiltering_nodlet was changed to "robot_top_3d_laser_link" as this link is where the LIDAR is connected.

3.6.2.6. KISS-ICP

As KISS-ICP is developed to be LIDAR odometry pipeline, and not a full SLAM system, it does not generate a global map by default, but by changing the range of the local map, it became the global map as the robot drove around. This was achieved by removing `RemovePointsFarFromLocation` function in `VoxelHasMap.cpp`. In addition to this, only modifications were made to `odometry.launch`, where the topic name and robot_odom were set according to the datasets.

3.6.2.7. Reconstructed maps

To export the reconstructed maps the pcl_ros package was used with the `pointcloud_to_pcd` function which saves all point clouds at a given topic to pcd format. As all algorithms published a full map to a topic, the input topic needed to be adjusted for each algorithm. As seen in the code snippet below. Then the last point cloud generated by each algorithm was considered as the reconstructed global map.

```
rosrun pcl_ros pointcloud_to_pcd input:=/<full-map-topic>
```

3.7. Calculation of density and volume

For both camera and LIDAR based approaches the same script was used to calculate density and volume. All algorithms that created a exportable map was either .ply or .pcd files, so by converting the .ply to .pcd the same script could be utilized. To calculate the densities and volumes of the resulting maps, the open3d [70] Python package was used. The density calculation was based on the number of points per cubed meter within the bounding box of the map.

4. RESULTS

This section presents the comprehensive evaluation conducted on various algorithms, leveraging both RTK-GNSS, LIDAR data and camera data to assess their performance across different environments and conditions. This analysis spans from dataset accuracy to trajectory and map reconstruction, with a focus on the precision and consistency of the SLAM systems under evaluation.

4.1. Dataset accuracy

To evaluate the performance of the algorithms on the dataset that was collected for this thesis, it is important to include the accuracy of the GNSS. This data is represented in Fig. 25 and in Fig. 26. In both plots, the longitude and latitude have identical values. From the plots, one can observe that within each plot the accuracy of the altitude is less accurate than longitude and latitude. It is also apparent that the performance in the orchard dataset is more accurate than the parking dataset.

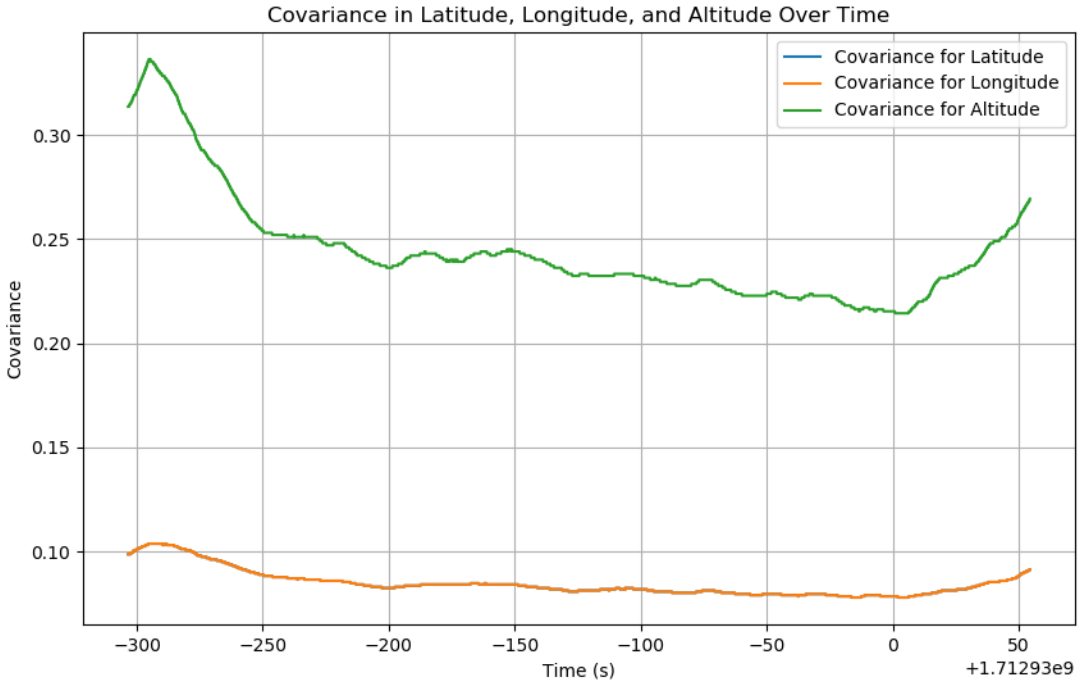


Fig. 25: Covariance of longitude, longitude and altitude for the **orchard** dataset. Unit of measurement is m^2. Longitude and latitude share covariance.

4.2. Camera

The evaluation of the results consists of both quantitative and qualitative methods, focusing on trajectory estimates and map reconstruction quality. The quantitative map reconstruction analysis involves looking

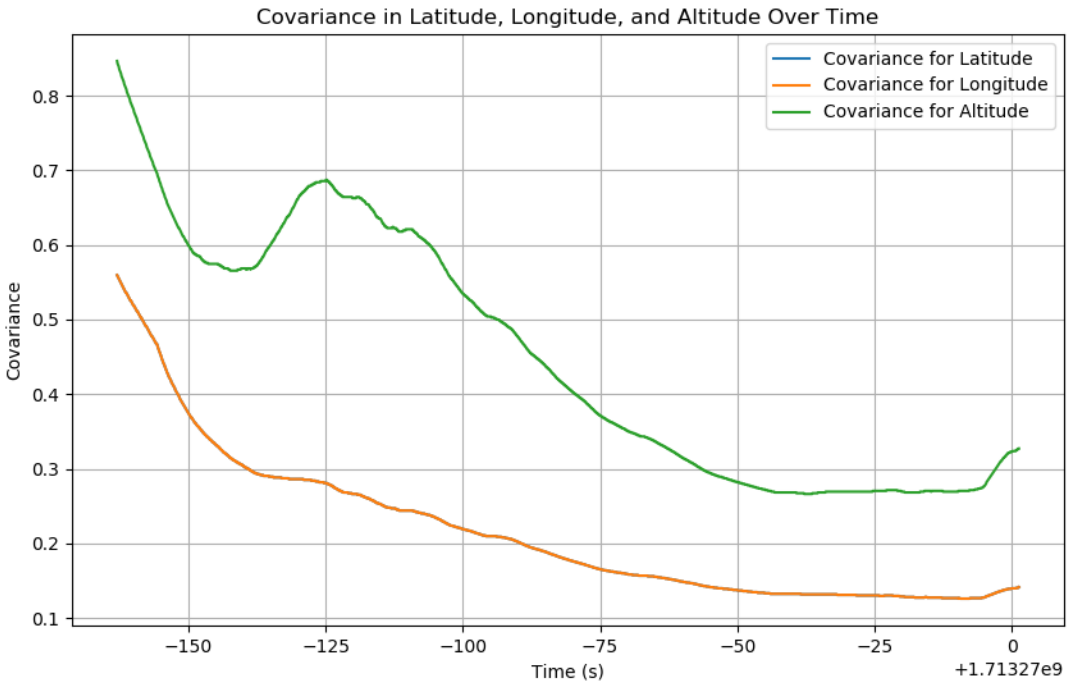


Fig. 26: Covariance of longitude, longitude and altitude for the **parking** dataset. Unit of measurement is in m^2, and Longitude and latitude share covariance.

at the file size, number of points, volume, and density, while the qualitative analysis concentrates on feature representation, detail, and consistency of the reconstructed maps. Trajectory analysis also employs both approaches, where the quantitative analysis involves utilizing ground truth data to measure the error metrics, whilst the qualitative analysis entails visual plot examination for qualitative coherence. The metrics evaluated include maximum error (Max), mean error (Mean), median error (Median), minimum error (Min), root mean square error (RMSE), and standard deviation (Std), with all units reported in meters.

Note; RESLAM did not have the ability to extract the reconstructed map into a .pcd or .ply file. Thus, **only** qualitative analysis for the reconstructed map was performed for RESLAM for **all** of the datasets.

4.2.1. TUM RGB-D Dataset

fr2_desk:

For the fr2_desk sequence, ORB-SLAM2 demonstrated remarkable performance with the lowest RMSE and standard deviation, indicating both high accuracy and precision. RTAB-Map, while exhibiting a larger mean error, maintained a relatively low standard deviation, suggesting consistency in its tracking albeit with a systematic bias. ManhattanSLAM and RESLAM, while having higher maximum errors, maintained competitive mean and median errors. The APE results for the fr2_desk sequence are shown in Table. 3.

The trajectories plotted against each other in both XZ and XYZ plots, is shown in Fig. 27 and 28. When evaluating the estimated trajectories qualitatively, a critical observation is the tight grouping of trajectories from different algorithms, indicating similar performance characteristics in the XZ plane. However, subtle deviations and occasional spread among the lines hint at varying degrees of drift or error accumulation over time. Exemplified, the trajectory estimate plotted for RTAB-Map closely resembles the performance in the error metrics, where it performed worst in almost all of them, and this seems valid when looking at the plotted trajectory compared to the others as it slightly deviates more than the others with respect to ground truth.

When looking at the reconstructed maps for this sequence, shown in Fig. 29, there are some clear differences. The reconstructed map by ORB-SLAM2 shows high detail in the feature-rich areas with a relatively clean and noise-free reconstruction. However, some sparsity is observed in less textured regions. RTAB-MAP's reconstruction exhibits a denser map with a good balance between detail and noise. The environmental structure is well-defined, although some minor artifacts are present. The map generated by RGBDSLAMv2 shows a dense point cloud reconstruction with detailed object reconstruction and surface contours. ManhattanSLAM's output presents a relatively dense reconstruction in feature-rich areas, such as the table with the objects, but produces a more sparse representation at surfaces where there are less features. RESLAM's map appears less dense compared to others, with sparse point clouds in regions of low texture. Despite this, the reconstruction maintains a coherent overall structure with fewer artifacts.

Algorithm	Max	Mean	Median	Min	RMSE	Std
ORB-SLAM2	**0.028737**	**0.007781**	**0.007625**	**0.000750**	**0.008328**	**0.002969**
RTAB-Map	0.175503	0.109263	0.104634	0.056007	0.112313	0.025998
RGBDSLAMv2	0.140237	0.087903	0.089168	0.026175	0.089630	0.017510
ManhattanSLAM	0.073054	0.027944	0.025050	0.000446	0.031950	0.015489
RESLAM	0.044868	0.018435	0.018180	0.002193	0.019431	0.006140

Table 3: APE w.r.t. translation part (m) with SE(3) Umeyama alignment [53] for fr2_desk sequence. **All units are in meters.**

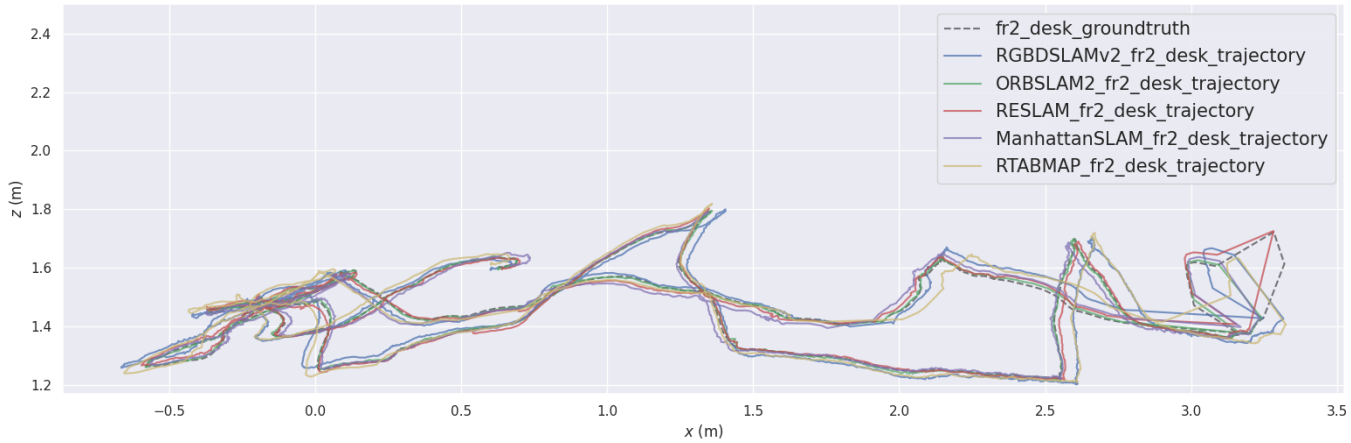


Fig. 27: XZ plot of trajectory estimate from each SLAM system for the fr2_desk sequence with ground truth alignment using the Umeyama method [53].

The statistics for each reconstructed map is shown in table 4. ORB-SLAM2, with a modest file size of 12.11 MB and 294,281 points, covered a volume of 1,700 cubic meters and exhibited a density of 173.24 points per cubic meter. RTAB-Map demonstrated a remarkable density of 25,115.96 points per cubic meter within a relatively small volume of 640 cubic meters, the smallest volume in the dataset, which correlated with its substantial file size of 42.67 MB. In contrast, RGBDSLAMv2, despite its larger file size of 165.73 MB and the highest number of points at 10,861,255, maintained a lower density of 4,853.71 points per cubic meter across a volume of 2,240 cubic meters. ManhattanSLAM had the lowest point density at 43.85 points per cubic meter in a volume of 1,440 cubic meters, accompanied by the smallest file size of 1.69 MB.

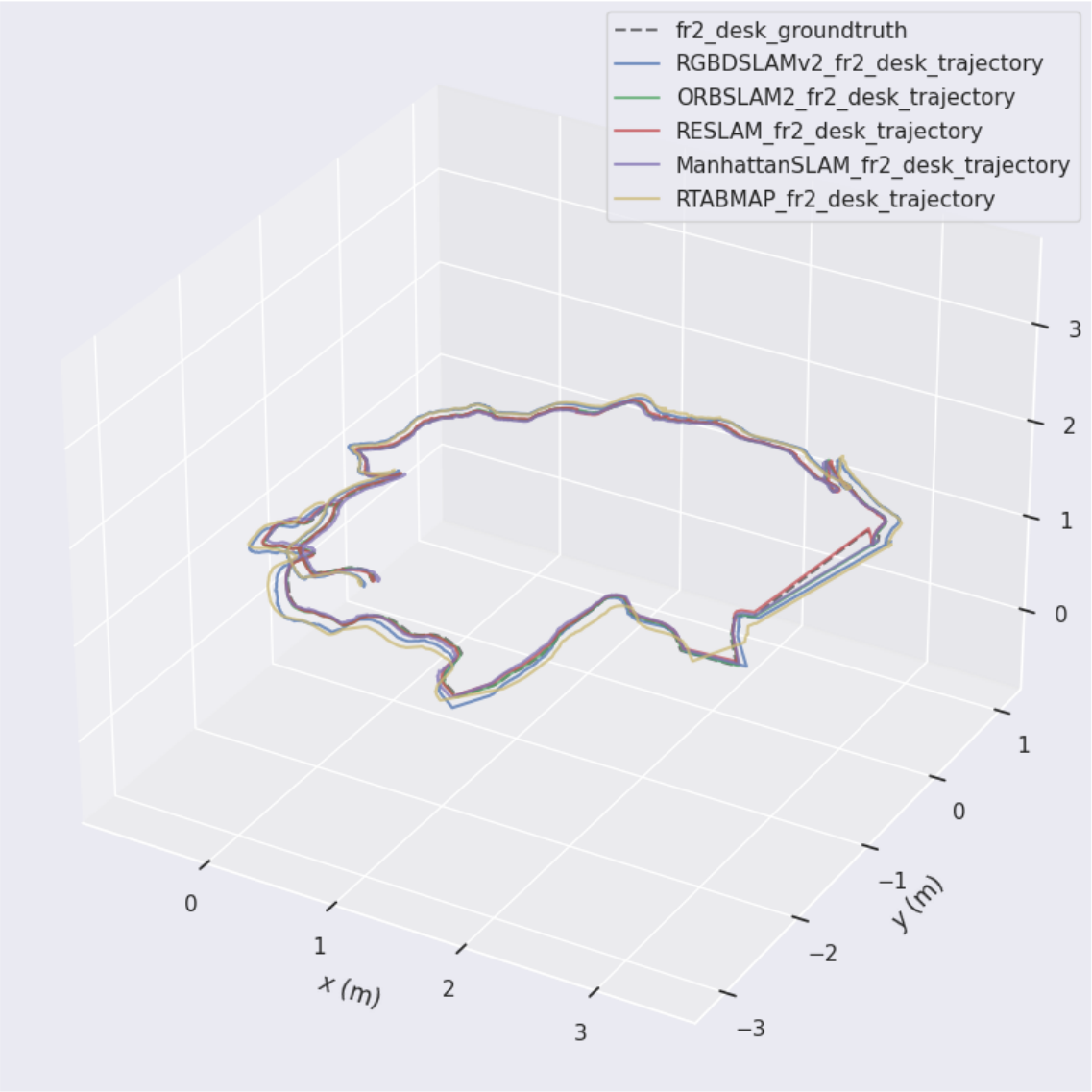


Fig. 28: XYZ plot of trajectory estimate from each SLAM system for the fr2_desk sequence with ground truth alignment using the Umeyama method [53].

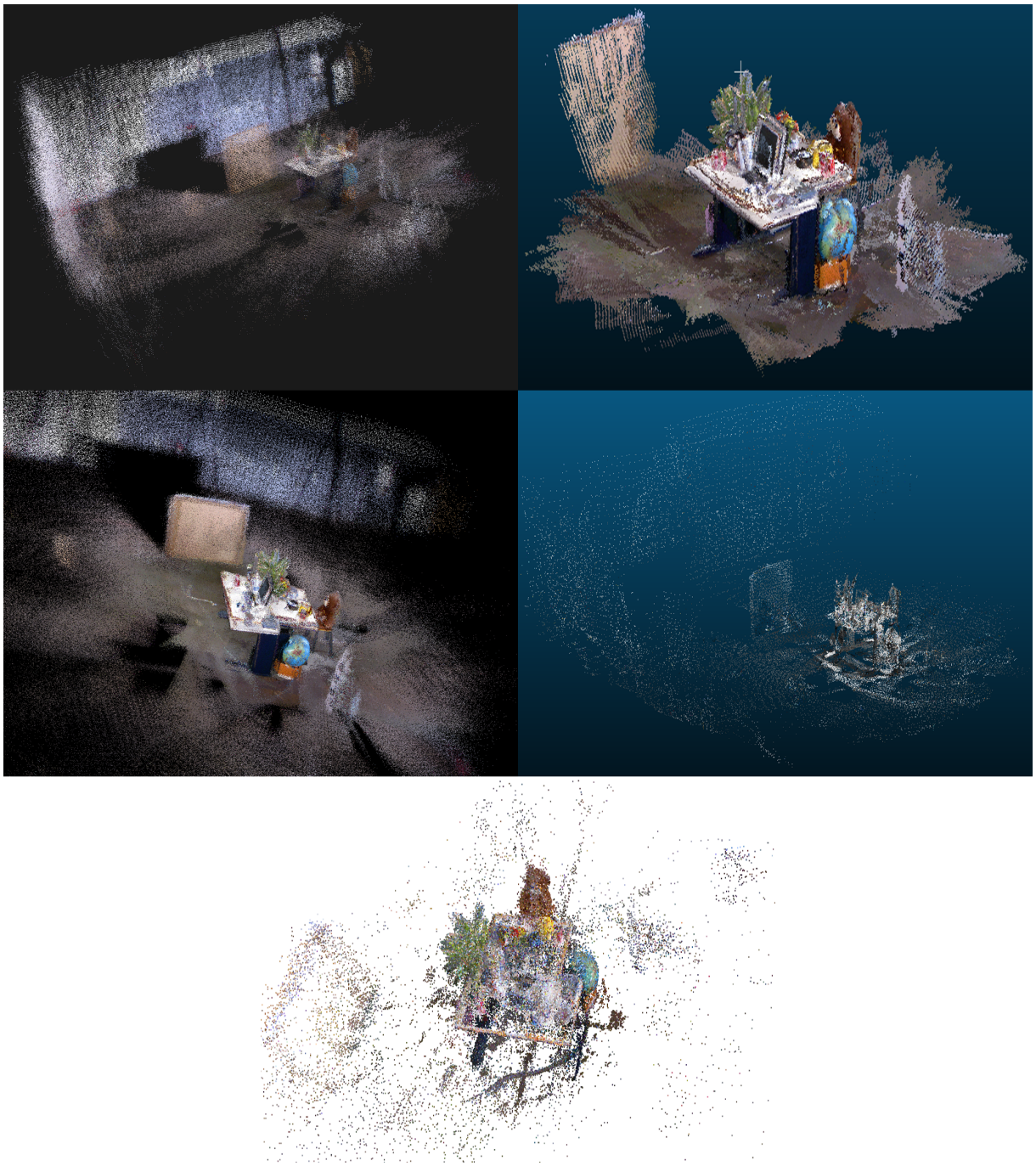


Fig. 29: Reconstructed maps for the fr2_desk sequence from each SLAM algorithm. Top-left: ORB-SLAM2, top-right: RTAB-Map, middle-left: RGBDSLAMv2, middle-right: ManhattanSLAM, bottom: RESLAM.

File Name	File Size (MB)	Number of Points	Volume (m^3)	Density (points/m^3)
ORB-SLAM2.pcd	12.11	294281	1.70e+03	173.24
RTAB-Map.pcd	42.67	1597853	0.64e+03	25115.96
RGBDSLAMv2.pcd	165.73	10861255	2.24e+03	4853.71
ManhattanSLAM.pcd	1.69	63338	1.44e+03	43.85
RESLAM.pcd	N/A	N/A	N/A	N/A

Table 4: Statistics for each reconstructed map for the fr2_desk sequence. For each map in point cloud data (pcd) format, there is file size, total number of points, volume of the bounding box, and density of the points in terms of volume.

fr2_desk_with_person:

Moving to the fr2_desk_with_person sequence, the results showcased an increase in error metrics for all algorithms when a person was interacting with the same environment. Despite this, ORB-SLAM2 managed to keep its RMSE low, although with a notable increase from the static environment. ManhattanSLAM showed an increased variance in performance with a higher RMSE. RTAB-Map and RGBDSLAMv2 had notably higher values in most of the error metrics for this sequence. In contrast, RESLAM achieved the lowest error metrics with the exception of the minimum error metric. The table showing the APE results for the fr2_desk_with_person sequence are shown in Table. 5.

The trajectories plotted against each other in both XZ and XYZ plots, is shown in Fig. 30 and 31. There are noticeable differences within the same environment when a dynamic object, in this case a person, is introduced. There are greater variances between the trajectories estimates to the ground truth, even for the better performing ones, signifying that dynamic objects cause the estimates to be worse. One more thing to note however, is that in this sequence there is no loop closure as it was in the fr2_desk sequence which is based in the same environment, this could also have a significant impact on the estimated trajectory for each of the SLAM systems, where the trajectory estimates are not as smooth as in the fr2_desk sequence.

The map for ORB-SLAM2 looks sparse and the features seem to be very localized around certain high-contrast areas. The person and desk are identifiable, but the overall structure lacks detail. The reconstructed map from RTAB-Map is very dense and provides good detail of the most feature-rich areas, but seems to not include a lot of the ground and walls from the entire sequence. RGBDSLAMv2's reconstruction looks very similar to ORB-SLAM2, but way denser. The details of the environment, including the desk and other objects, are much clearer. The reconstruction for ManhattanSLAM and RESLAM appears more sparse than the others with respect to less feature-rich areas such as the ground and walls. The reconstructed maps are shown in Fig. 32.

The statistics for each reconstructed map is shown in table 6. ORB-SLAM2 had a relatively small file size of 8.37 MB and contained 203,610 points within a volume of approximately 2010 cubic meters, resulting

Algorithm	Max	Mean	Median	Min	RMSE	Std
ORB-SLAM2	0.068307	0.05538	0.05085	**0.000307**	0.06294	0.02990
RTAB-Map	0.146268	0.107228	0.104088	0.078417	0.108187	0.014371
RGBDSLAMv2	0.140278	0.076161	0.077577	0.018702	0.079397	0.022436
ManhattanSLAM	0.089529	0.034721	0.035829	0.007985	0.038494	0.016620
RESLAM	**0.065306**	**0.030606**	**0.030592**	0.002822	**0.032225**	**0.011596**

Table 5: APE w.r.t. translation part (m) with SE(3) Umeyama alignment [53] for fr2_desk_with_person sequence. **All units are in meters.**

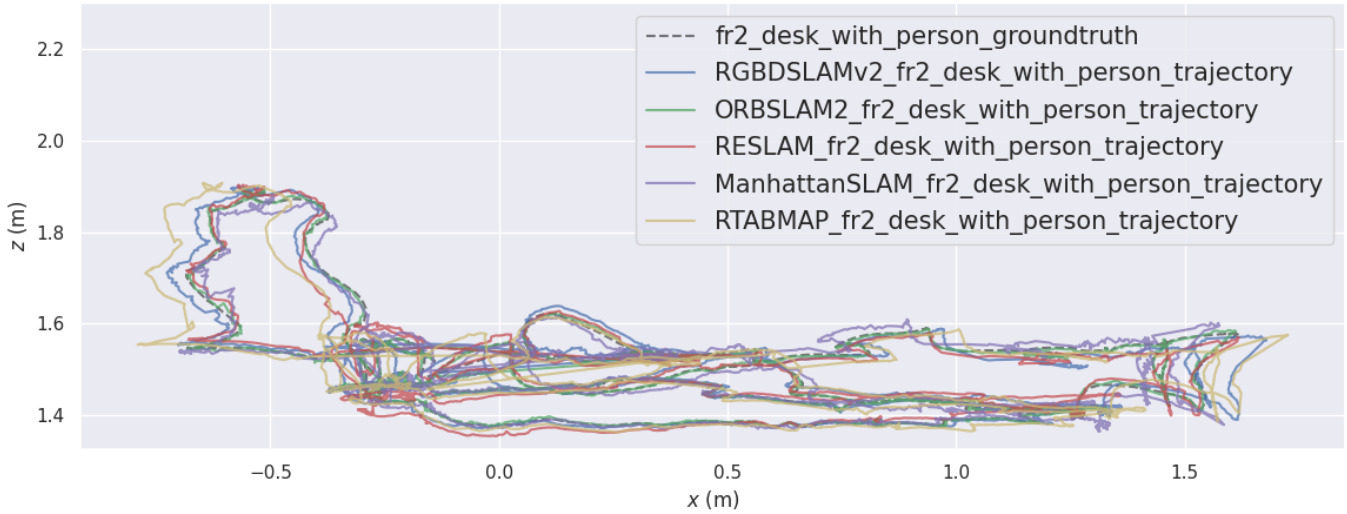


Fig. 30: XZ plot of trajectory estimate from each SLAM system for the fr2_desk_with_person sequence with ground truth alignment using the Umeyama method [53].

in a density of 101.16 points per cubic meter. RTAB-Map had a smaller bounding box volume of 83 cubic meters, but packed a significantly higher number of points, reflecting a dense cluster of 22,076.35 points per cubic meter. Moreover, RGBDSLAMv2, the largest file at 214.56 MB, contained the highest number of points, spread over a volume of 2740 cubic meters, yielding a density of 5122.83 points per cubic meter. Finally, ManhattanSLAM presented the lowest density of 45.91 points per cubic meter, with 79,058 points across a 1720 cubic meter volume.

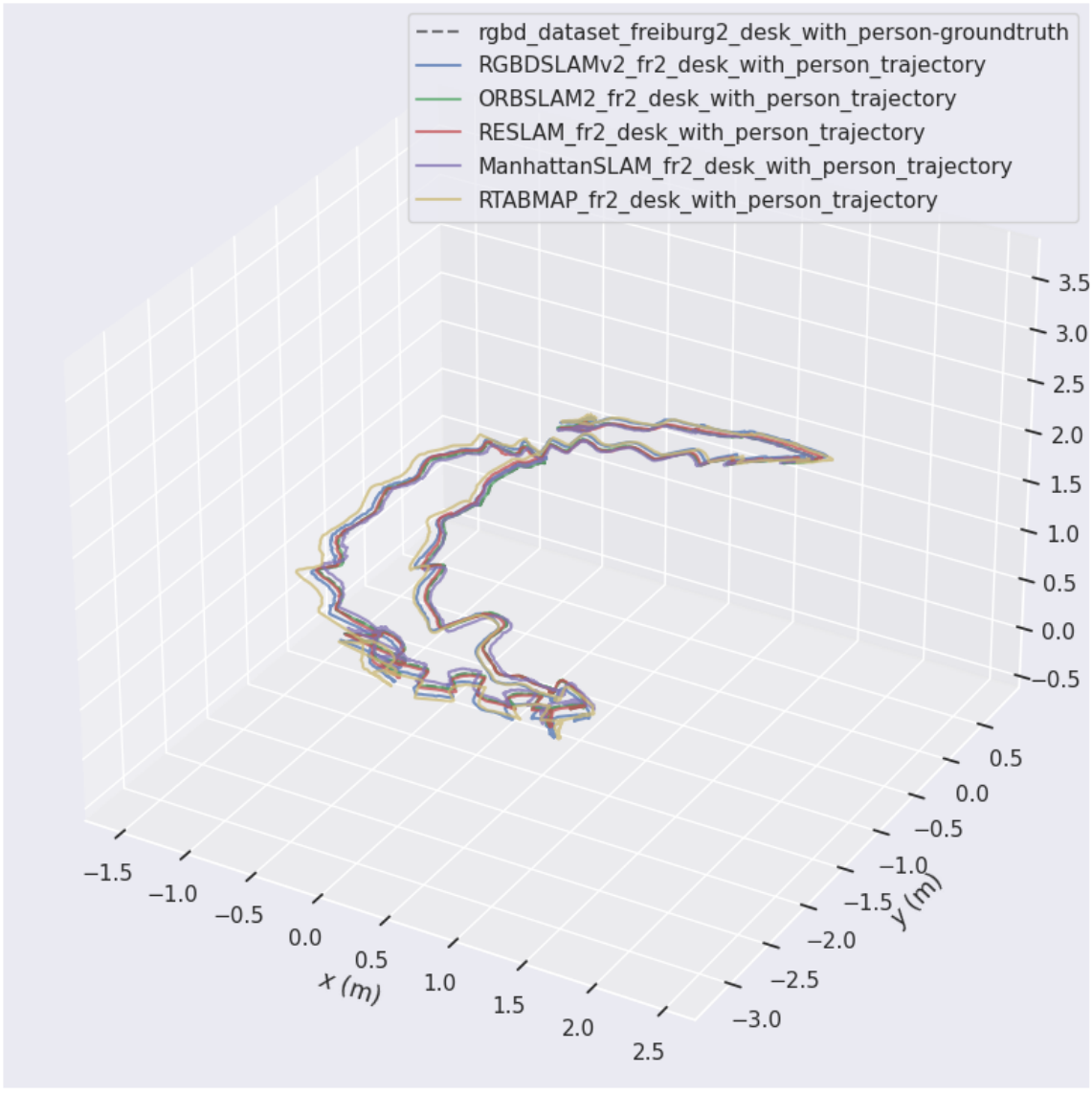


Fig. 31: XYZ plot of trajectory estimate from each SLAM system for the fr2_desk_with_person sequence with ground truth alignment using the Umeyama method [53].

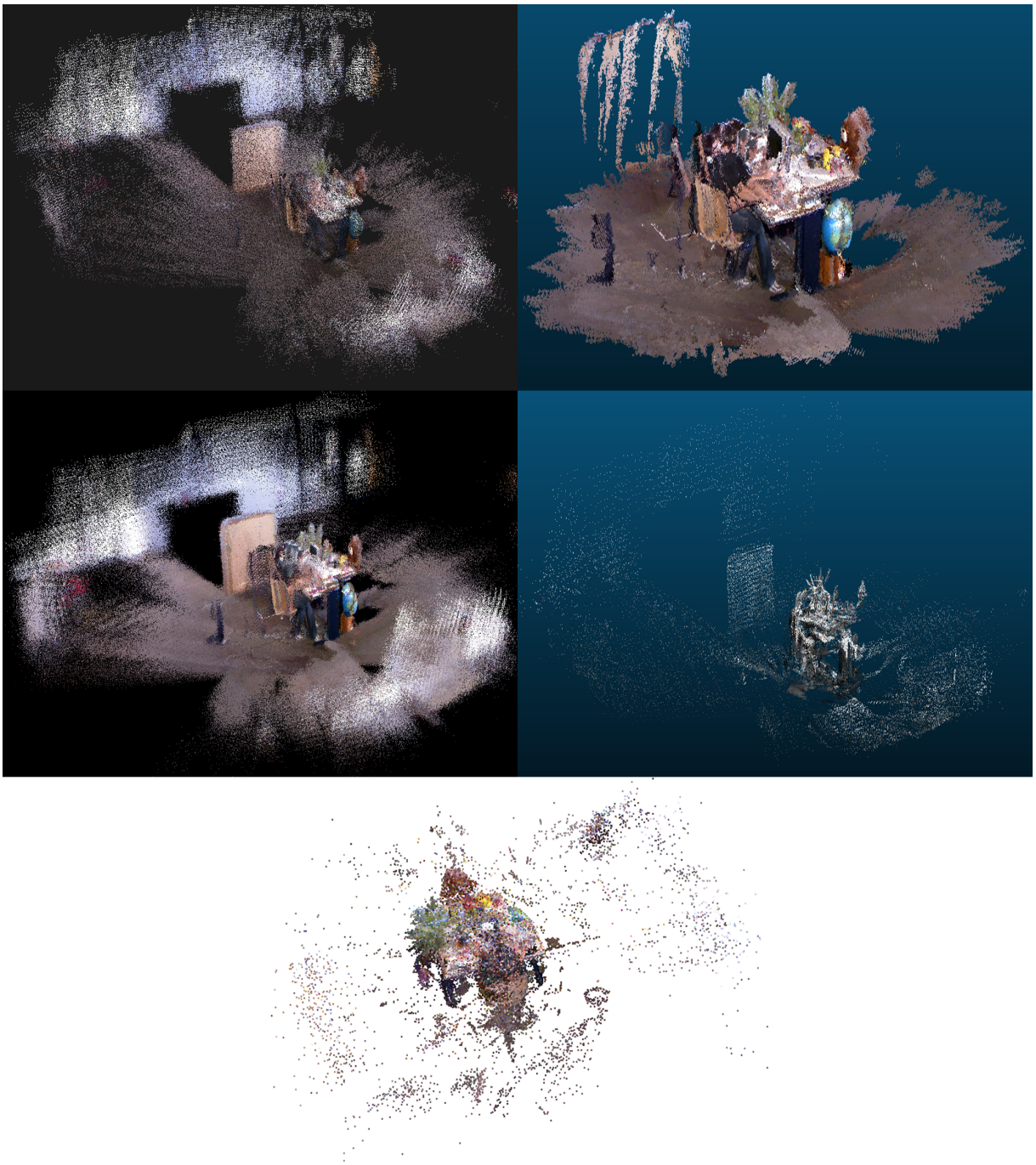


Fig. 32: Reconstructed maps for the fr2_desk_with_person sequence from each SLAM algorithm. Top-left: ORB-SLAM2, top-right: RTAB-Map, middle-left: RGBDSLAMv2, middle-right: ManhattanSLAM, bottom: RESLAM.

File Name	File Size (MB)	Number of Points	Volume (m^3)	Density (points/m^3)
ORB-SLAM2.pcd	8.37	203610	2.01e+03	101.16
RTAB-Map.pcd	48.75	1825496	0.083e+03	22076.35
RGBDSLAMv2.pcd	214.56	14060916	2.74e+03	5122.83
ManhattanSLAM.pcd	2.11	79058	1.72e+03	45.91
RESLAM.pcd	N/A	N/A	N/A	N/A

Table 6: Statistics for each reconstructed map for the fr2_desk_with_person sequence. For each map in point cloud data (pcd) format, there is file size, total number of points, volume of the bounding box, and density of the points in terms of volume.

fr3_walking_xyz:

The fr3_walking_xyz sequence results indicate a significant increase in localization errors for most algorithms, likely due to the complexity introduced by rapid 3D motion while simultaneously incorporating dynamic objects. ORB-SLAM2, while showing a decent increase in its RMSE and max error, still maintained a relatively lower mean error compared to ManhattanSLAM and RESLAM. RGBDSLAMv2 demonstrated a low RMSE, potentially indicating a more effective strategy for dealing with full 3D motion. RTAB-MAP also had a substantial increase in terms of performance for this sequence compared to the previous ones. The table showing the APE results for the fr3_walking_xyz sequence are shown in Table. 7.

The trajectories plotted against each other in both XZ and XYZ plots, is shown in Fig. 33 and 34. Upon inspection of the trajectory plots, it can be clearly seen that the estimated trajectories for this sequence is highly irregular and erratic. When looking at how these SLAM systems performed during run-time, it could be noticed that most of them struggled when one of the people are moving in the scene while simultaneously the camera is being moved along the x-axis. It could be made the case for that the only SLAM systems that performed well in this sequence based on the trajectory plot, was RGBDSLAMv2 and RTAB-Map. The others had great deviations from ground truth.

The reconstructed maps for this sequence has less detailed features than the two previous sequences with the desk. The reconstructions for ORB-SLAM2 and RGBDSLAMv2 look quite similar except that ORB-SLAM2's reconstruction is a bit more sparse and includes more information about areas where there are not many features as can be seen at the top of the reconstruction for both of them compared. RTAB-Map's reconstruction got a lot of detail from the main part of the scene which was the people and the desk, but did not include information about the rest of the area. ManhattanSLAM and RESLAM give relatively sparse reconstructions compared to RGBDSLAMv2 and RTAB-Map and details are not as clear in these reconstructions. The reconstructed maps for this sequence is shown in Fig. 35.

The statistics for each reconstructed map is shown in table 8. ORB-SLAM2, with a file size of 15.77 MB

Algorithm	Max	Mean	Median	Min	RMSE	Std
ORB-SLAM2	0.726983	0.332119	0.323284	0.015219	0.392336	0.208864
RTAB-Map	0.556849	0.097965	0.075098	0.012696	0.129375	0.084503
RGBDSLAMv2	**0.111939**	**0.027264**	**0.022928**	**0.002296**	**0.032240**	**0.017206**
ManhattanSLAM	1.563303	0.517755	0.411244	0.030480	0.626333	0.352453
RESLAM	1.513727	0.832047	0.819912	0.206197	0.906516	0.359817

Table 7: APE w.r.t. translation part (m) with SE(3) Umeyama alignment [53] for fr3_walking_xyz sequence. **All units are in meters.**

and 386,068 points, covers a volume of approximately 1,290 cubic meters, resulting in a density of around 299 points per cubic meter. RTAB-Map, although smaller in volume with about 54 cubic meters, provides a much higher density of 16,519.29 points per cubic meter, facilitated by a larger file size of 23.84 MB and a total of 892,609 points. Furthermore, the data from RGBDSLAMv2 and ManhattanSLAM further underline these discrepancies. The RGBDSLAMv2.pcd, with the largest file size at 36.25 MB and the highest point count of 2,375,609, shows a volume similar to that of ORB-SLAM2, but with a significantly higher density of 2,064.36 points per cubic meter. Additionally, ManhattanSLAM, despite having the smallest file size of 1.55 MB and the lowest number of points, spans nearly the same volume as ORB-SLAM2, but with an exceedingly low density of 54.64 points per cubic meter.

File Name	File Size (MB)	Number of Points	Volume (m^3)	Density (points/m^3)
ORB-SLAM2.pcd	15.77	386068	1.29e+03	299.03
RTAB-Map.pcd	23.84	892609	0.054e+03	16519.29
RGBDSLAMv2.pcd	36.25	2375609	1.15e+03	2064.36
ManhattanSLAM.pcd	1.55	58103	1.06e+03	54.64
RESLAM.pcd	N/A	N/A	N/A	N/A

Table 8: Statistics for each reconstructed map for the fr3_walking_xyz sequence. For each map in point cloud data (pcd) format, there is file size, total number of points, volume of the bounding box, and density of the points in terms of volume.

fr3_sitting_static:

Finally, in the fr3_sitting_static sequence, we see a return to lower error metrics across all algorithms, which makes sense since 3D motion and quickly moving dynamic objects are out of the picture. Thus, while not surprising, every algorithm performed well and there was not an exceptionally outstanding performer even though RGBDSLAMv2 performed the best. The low standard deviations across the board suggest that the static nature of the sequence minimized the influence of external variables on algorithm performance, such as movement or occlusion. The table showing the APE results for the fr3_sitting_static sequence are shown in Table. 9.

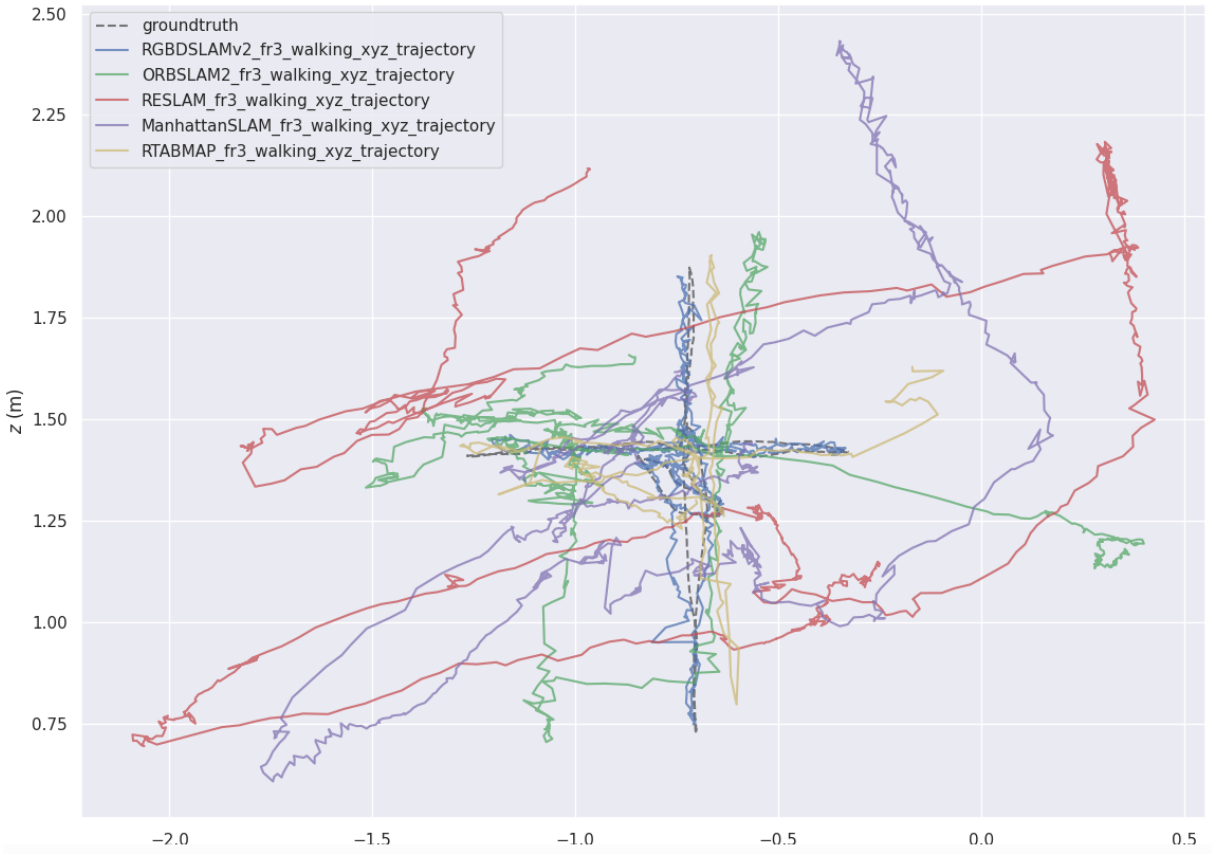


Fig. 33: XZ plot of trajectory estimate from each SLAM system for the fr3_walking_xyz sequence with ground truth alignment using the Umeyama method [53].

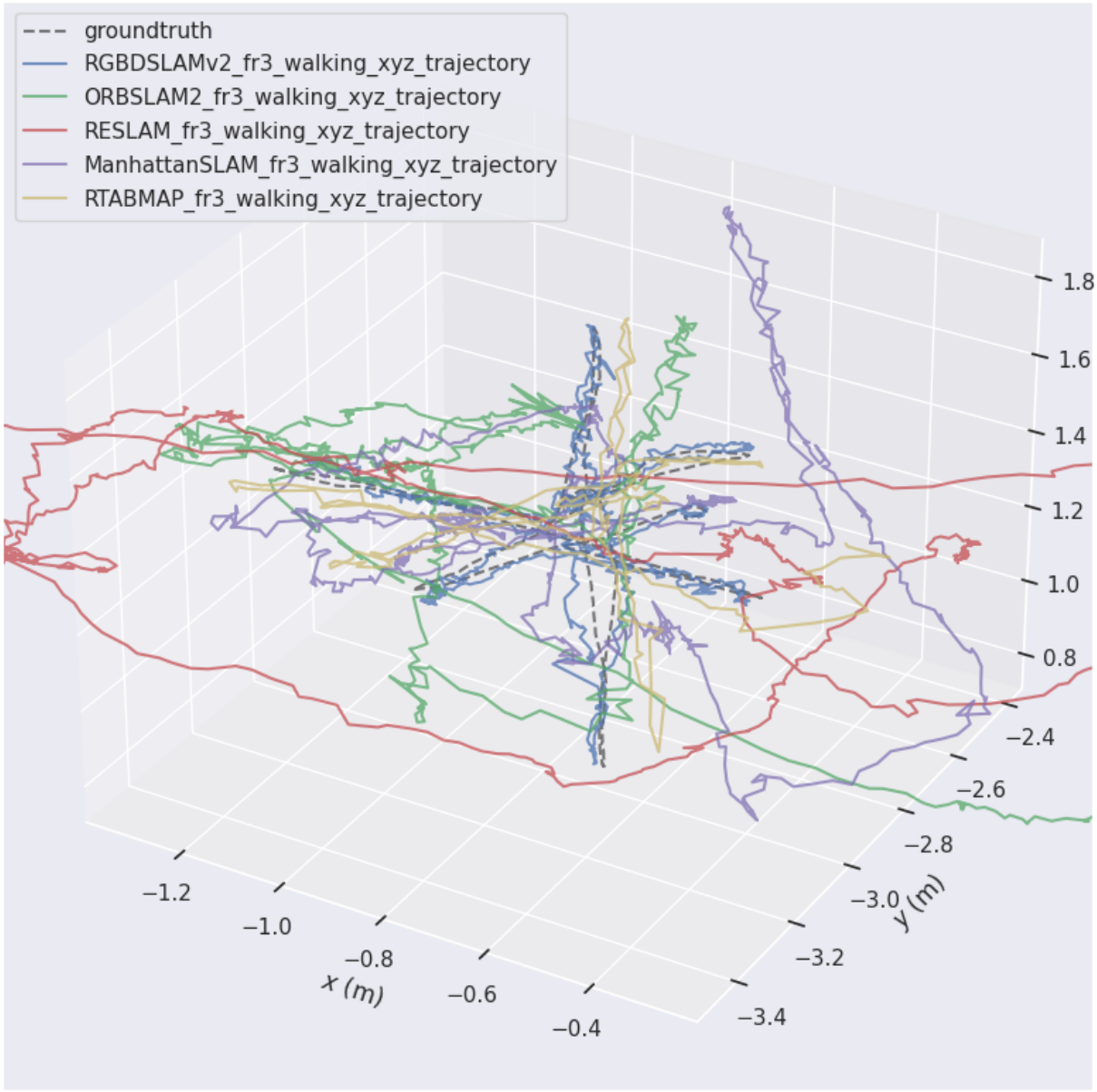


Fig. 34: XYZ plot of trajectory estimate from each SLAM system for the fr3_walking_xyz sequence with ground truth alignment using the Umeyama method [53].

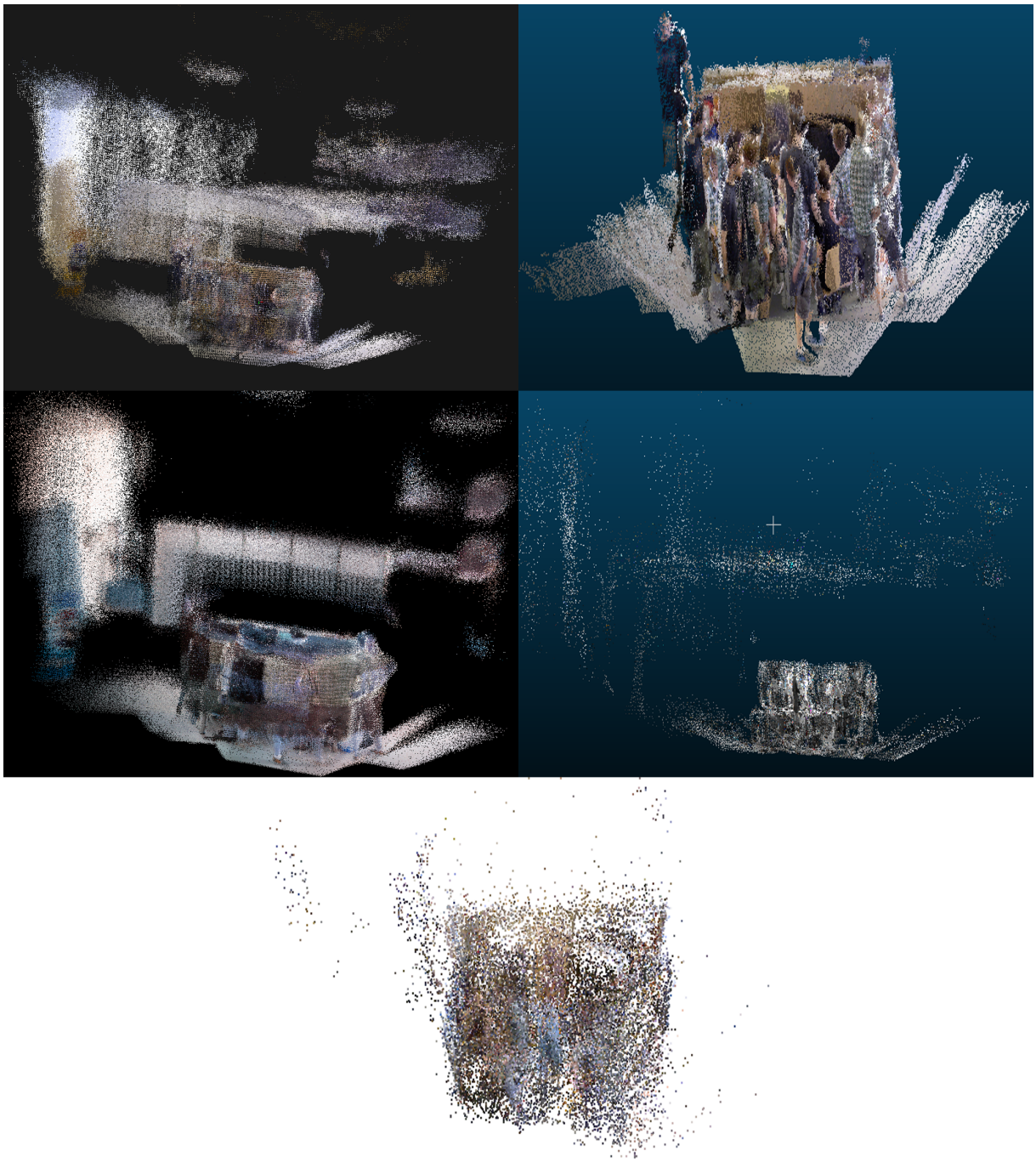


Fig. 35: Reconstructed maps for the fr3_walking_xyz sequence from each SLAM algorithm. Top-left: ORB-SLAM2, top-right: RTAB-Map, middle-left: RGBDSLAMv2, middle-right: ManhattanSLAM, bottom: RESLAM.

Algorithm	Max	Mean	Median	Min	RMSE	Std
ORB-SLAM2	0.037843	0.007270	0.006570	**0.000438**	0.008323	0.004052
RTAB-Map	0.031495	0.008061	0.007381	0.001178	0.009219	0.004473
RGBDSLAMv2	**0.027057**	**0.006192**	**0.005557**	0.000942	**0.006976**	**0.003213**
ManhattanSLAM	0.035891	0.008120	0.007755	0.000942	0.008978	0.003828
RESLAM	0.040538	0.007428	0.006483	0.000620	0.008962	0.005014

Table 9: APE w.r.t. translation part (m) with SE(3) Umeyama alignment [53] for fr3_sitting_static sequence. **All units are in meters.**

The trajectory plots for this sequence are not that easy to analyze qualitatively as the estimated trajectories are very erratic and covers the ground truth trajectory quite a bit. However, they all seem to have the ”same” pattern across the entire plot which makes sense given the fact that they all performed well with respect to error metrics. The trajectories plotted against each other in both XZ and XYZ plots, is shown in Fig. 36 and Fig. 37.

As this sequence had no movement alongside any axis, the reconstruction for some of the SLAM systems did not include as much information as the one in the previous sequence, given that they were recorded in the same area. ORB-SLAM2 and RGBDSLAMv2 are still looking quite similar, but with the exception that ORB-SLAM2's reconstructed is yet again more sparse. RTAB-Map had a quite similar reconstruction as the one in the previous sequence, but with less artifacts and noise. ManhattanSLAM, obtained a more dense reconstruction in this sequence compared to the previous one. Finally, it seems that RESLAM had issues with capturing details, even in the feature-rich areas, resulting in a sparse, low-detail reconstruction. The reconstructed maps for this sequence is shown in Fig. 38.

The statistics for each reconstructed map is shown in table 10. ORB-SLAM2 shows a modest file size of 1.38 MB with 33,658 points in a 460 cubic meter space, rendering a point density of 72.57 points per cubic meter. The RTAB-Map, provides a file size of 10.65 MB, including about 398,913 points, confined within a notably smaller volumetric space of 23 cubic meters. This yields an exceptionally high point density of 17,456.85 points per cubic meter. The RGBDSLAMv2 file, with its considerably larger file size of 33.89 MB, contains over 2.2 million points within a volume of 730 cubic meters, resulting in a lower density of 3,057.83 points per cubic meter. In contrast, ManhattanSLAM, with the smallest file size of 0.83 MB, captures 30,915 points within 400 cubic meters, achieving a density of 76.49 points per cubic meter.

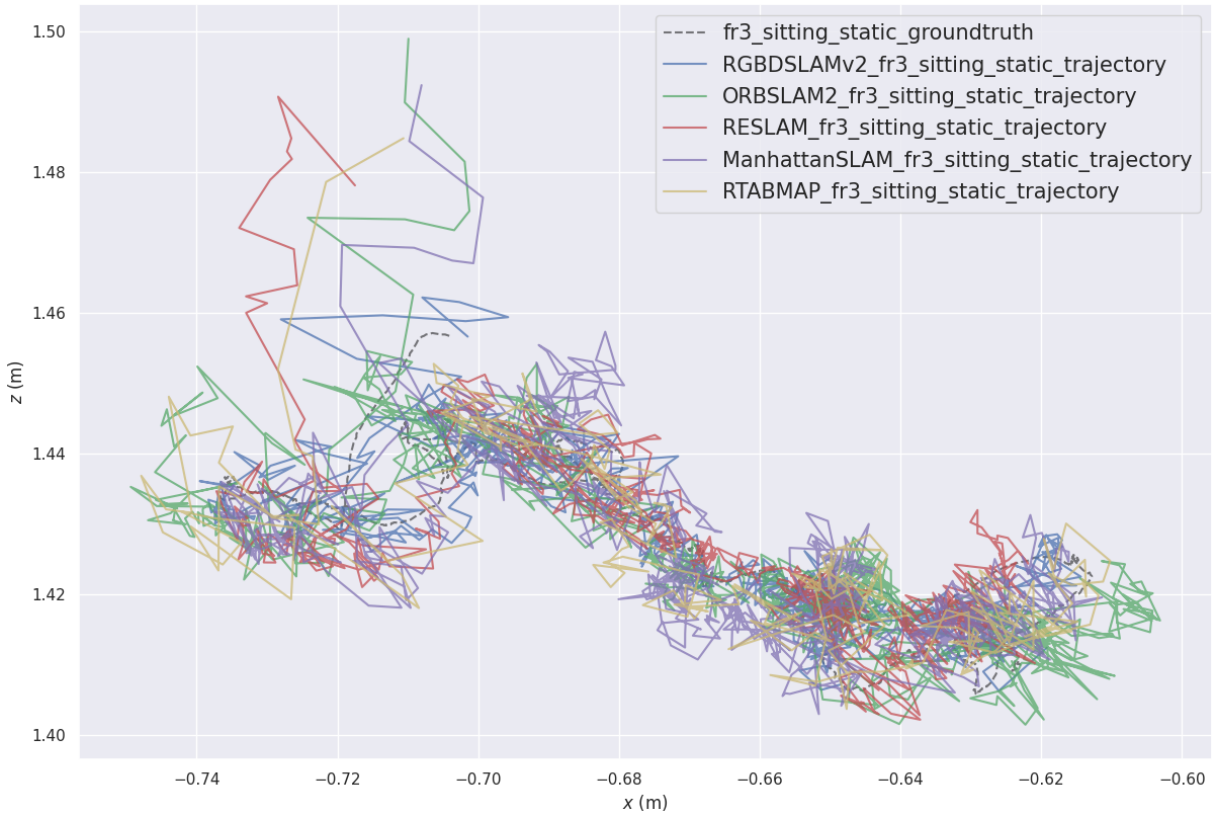


Fig. 36: XZ plot of trajectory estimate from each SLAM system for the fr3_sitting_static sequence with ground truth alignment using the Umeyama method [53].

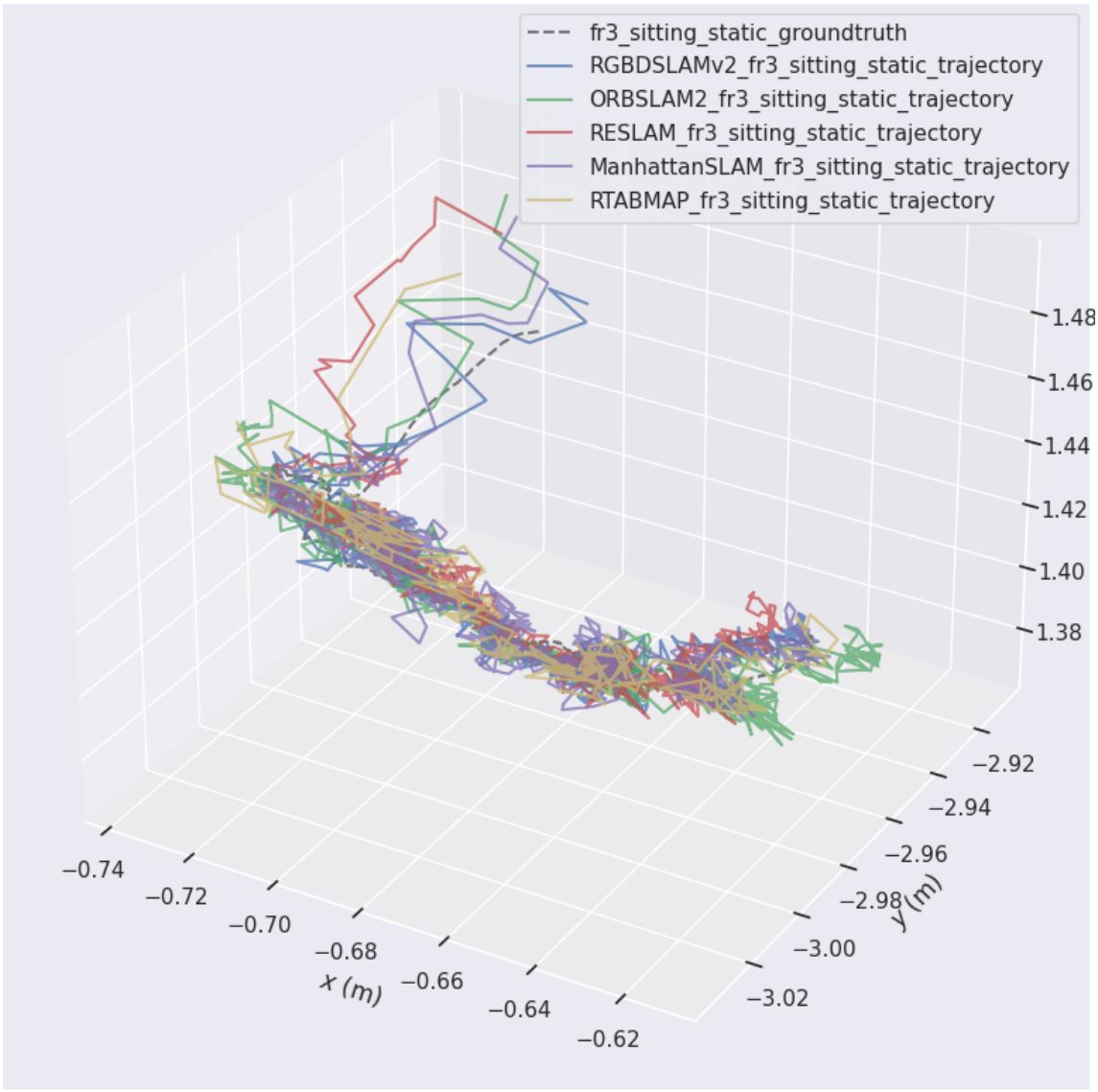


Fig. 37: XYZ plot of trajectory estimate from each SLAM system for the fr3_sitting_static sequence with ground truth alignment using the Umeyama method [53].

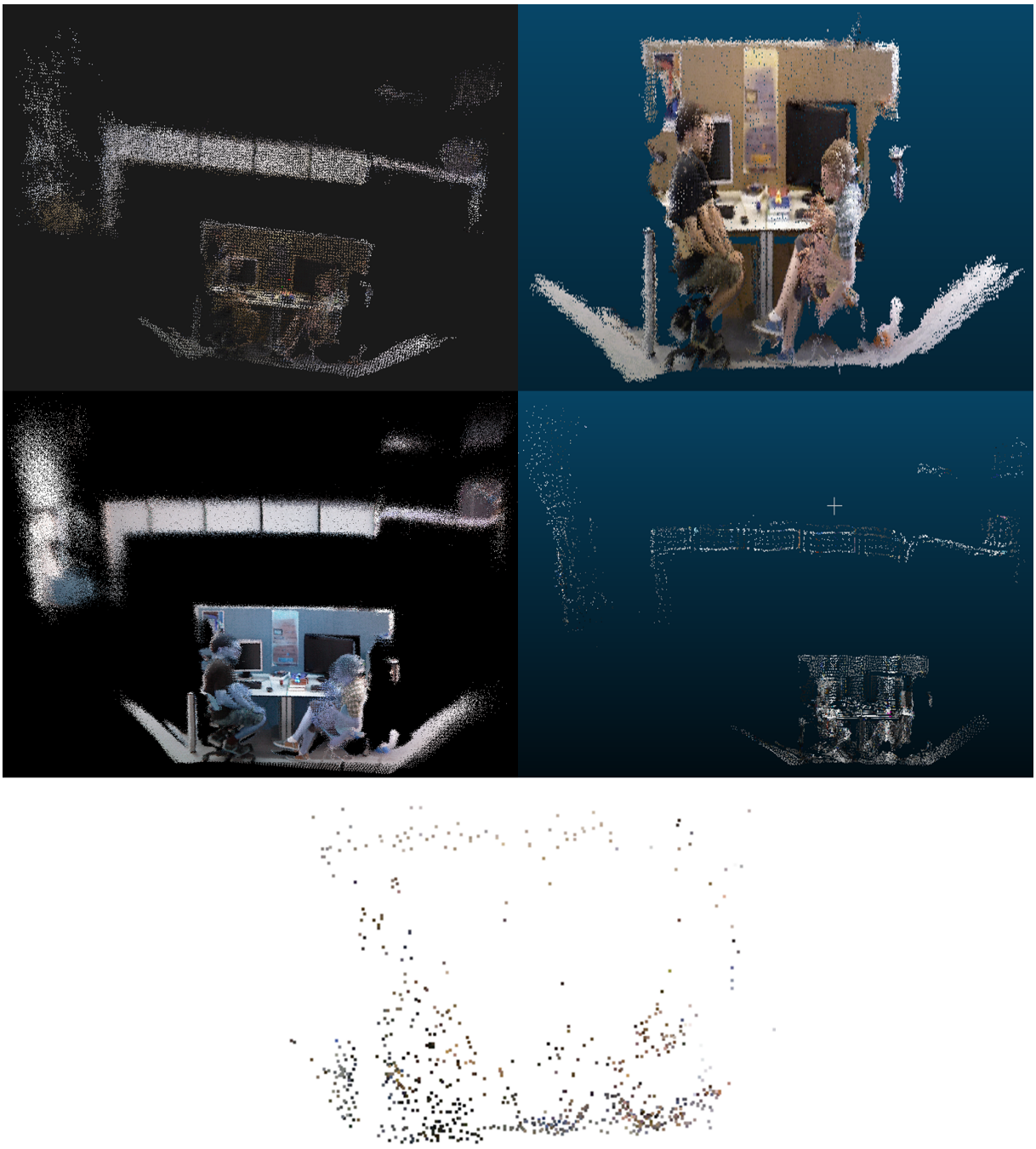


Fig. 38: Reconstructed maps for the fr3_sitting_static sequence from each SLAM algorithm. Top-left: ORB-SLAM2, top-right: RTAB-Map, middle-left: RGBDSLAMv2, middle-right: ManhattanSLAM, bottom: RESLAM.

File Name	File Size (MB)	Number of Points	Volume (m^3)	Density (points/m^3)
ORB-SLAM2.pcd	1.38	33658	0.46e+03	72.57
RTAB-Map.pcd	10.65	398913	0.023e+03	17456.85
RGBDSLAMv2.pcd	33.89	2220854	0.73e+03	3057.83
ManhattanSLAM.pcd	0.83	30915	0.40e+03	76.49
RESLAM.pcd	N/A	N/A	N/A	N/A

Table 10: Statistics for each reconstructed map for the fr3_sitting_static sequence. For each map in point cloud data (pcd) format, there is file size, total number of points, volume of the bounding box, and density of the points in terms of volume.

4.2.2. Orchard dataset

For the orchard dataset, ORB-SLAM2 demonstrated outstanding performance across several metrics for the orchard dataset, notably achieving the lowest mean error, median error, RMSE, and standard deviation. RTAB-Map, while slightly outperforming ORB-SLAM2 in terms of maximum and minimum error, showed slightly higher values in mean and median errors, as well as RMSE. RGBDSLAMv2, on the other hand, exhibited substantially higher errors across all metrics, with a particularly notable maximum error of 15.279077 m. The mean and RMSE values exceeded 8 m, significantly higher than those of the other algorithms. ManhattanSLAM showed moderate performance with its metrics lying between the best and the worst performer. RESLAM, marked with an asterisk, indicates partial evaluation due to memory issues during runtime, covering approximately half of the dataset. Running RESLAM on half of the dataset consumed 10GB of RAM which was the maximum amount of RAM dedicated to the VM. With this limitation in mind, RESLAM achieved quite remarkable results. However, since it was not ran on the entire dataset, it is not possible to conclude that it would outperform the others on the entire dataset. On the other hand, if it would continue to perform the way it did for the first half, ORB-SLAM2 would have a quite challenging competitor at hand for this dataset. The table showing the APE results for the orchard dataset are shown in Table. 11.

The most noticeable characteristic of the plots from the orchard dataset, shown in Fig. 39 and Fig. 40, is the trajectory estimate from RGBDSLAMv2 which is relatively poor compared to the other ones. While not a surprise from the error metrics, it seems that it struggles during the sharp U-turn, which could lead to accumulative errors. The rest of the SLAM systems seem to have very closely related trajectory estimates for this dataset and there is not a totally clear distinction. On the other hand, looking at the estimated trajectory of RESLAM, it seems to be the best performing and as noted earlier, if it was able to run the whole dataset, the trajectory estimate could very well be almost flawless except for the U-turn where all of the SLAM systems struggled.

The reconstructed maps for this sequence is shown in Fig. 41. ORB-SLAM2, RTAB-Map and RESLAM

displayed a quite good reconstruction of the grass and trees, whilst RGBDSLAMv2 seemed to suffer quite a bit here with the plot not yielding much information about the orchard. ManhattanSLAM's reconstruction was not entirely disappointing, but not impressive either, the reconstruction of the grass and tress where the robot actually drove is in the middle of the plot. Meanwhile, the reconstruction at the outside of this orchard is from the building at the top and bottom of the orchard, which is not necessarily the point of interest in the reconstruction.

The statistics for each reconstructed map is shown in table 12. The ORB-SLAM2 file, at a size of 98.61 MB, contains approximately 2.4 million points within a bounding box volume of 11,600 cubic meters, resulting in a point density of 207.03 points per cubic meter. The RTAB-Map file, at a larger file size of 281.23 MB, also displays a remarkably higher density of 3732.91 points per cubic meter within a significantly smaller volume of 2,820 cubic meters. Further analysis shows that the RGBDSLAMv2 file, the largest among the set with a file size of 670.15 MB, contains an extremely high number of points at approximately 43.9 million, while also covering an extensively larger volume of 3.19 million cubic meters, which results in a substantially lower density of 13.77 points per cubic meter. In contrast to this, ManhattanSLAM, with a minimal file size of 10.41 MB, includes around 390,021 points distributed over a volume of 1.12 million cubic meters, which results in an extremely low density of 0.35 points per cubic meter.

Algorithm	Max	Mean	Median	Min	RMSE	Std
ORB-SLAM2	1.876483	**1.203415**	**1.180617**	0.621010	**1.225237**	**0.230212**
RTAB-Map	**1.854631**	1.278669	1.310110	**0.272796**	1.303255	0.251950
RGBDSLAMv2	15.279077	8.678303	9.043291	2.604970	9.198417	3.049252
ManhattanSLAM	3.324453	1.710893	1.549815	0.569914	1.793658	0.538568
RESLAM*	2.351481*	0.454535*	0.234107*	0.066266*	0.669685*	0.491809*

Table 11: Absolute positional error w.r.t. translation part (m) with Sim(3) Umeyama alignment [53] for the orchard dataset. **All units are in meters.** *RESLAM evaluation was only taken on about half of the dataset due to issues with memory during run-time of the SLAM system.

File Name	File Size (MB)	Number of Points	Volume (m^3)	Density (points/m^3)
ORB-SLAM2.pcd	98.61	2403870	1.16e+04	207.03
RTAB-Map.pcd	281.23	10531943	2.82e+03	3732.91
RGBDSLAMv2.pcd	670.15	43918894	3.19e+06	13.77
ManhattanSLAM.pcd	10.41	390021	1.12e+06	0.35
RESLAM.pcd	N/A	N/A	N/A	N/A

Table 12: Statistics for each reconstructed map for the orchard dataset. For each map in point cloud data (pcd) format, there is file size, total number of points, volume of the bounding box, and density of the points in terms of volume.

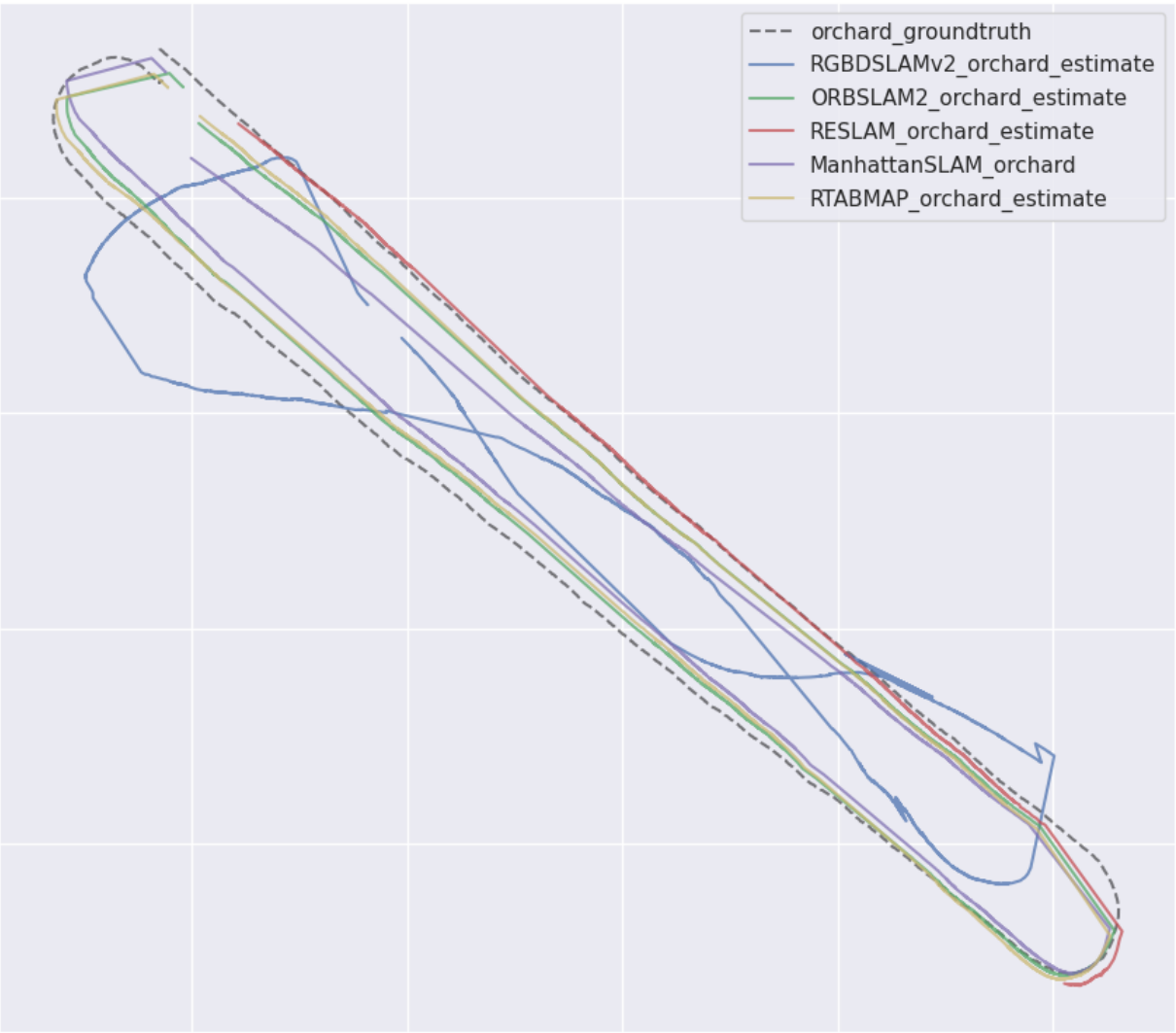


Fig. 39: XY plot of trajectory estimate from each SLAM system for the orchard dataset with ground truth alignment using the Umeyama method [53].

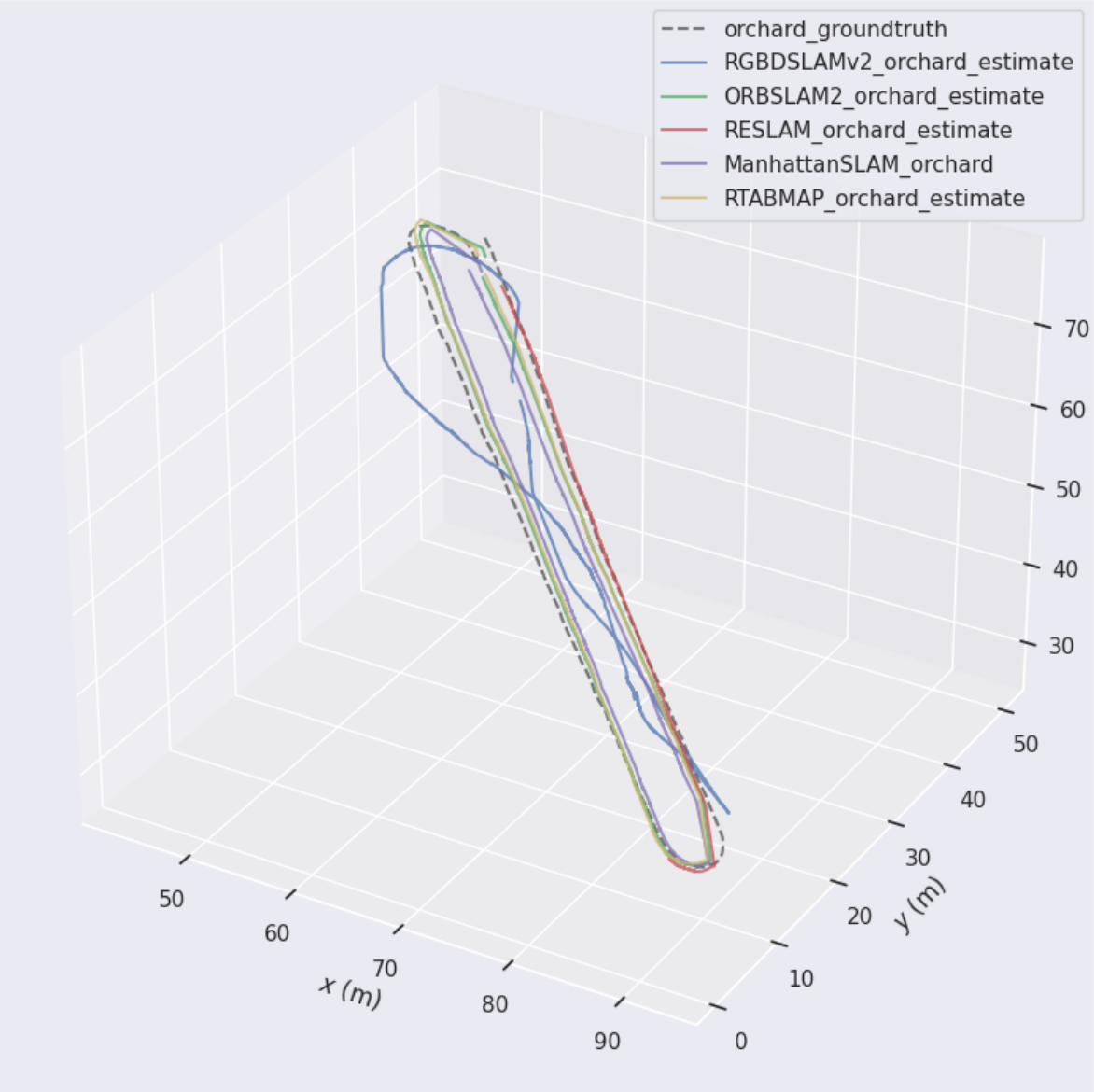


Fig. 40: XYZ plot of trajectory estimate from each SLAM system for the orchard dataset with ground truth alignment using the Umeyama method [53].

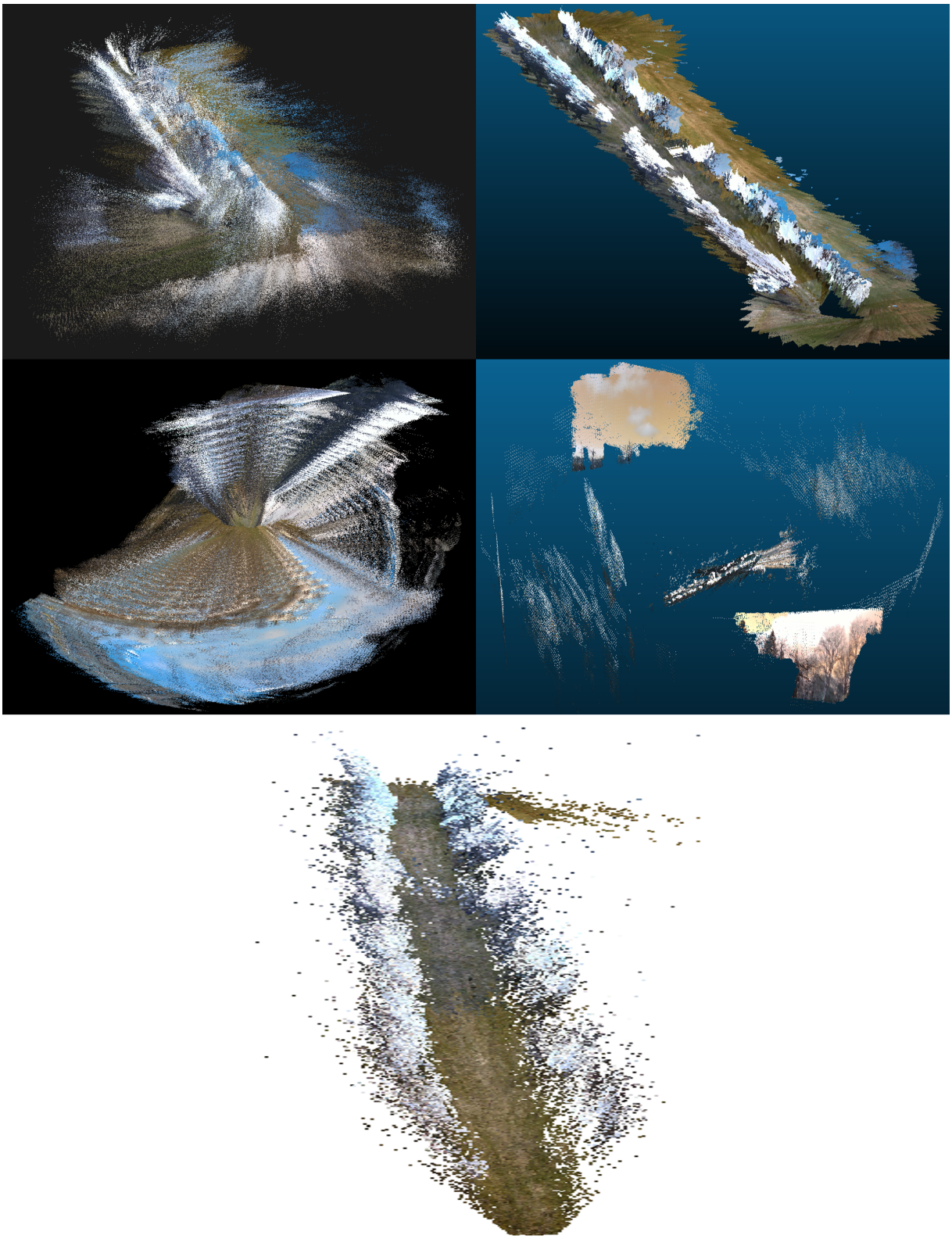


Fig. 41: Reconstructed maps for the orchard dataset from each SLAM algorithm. Top-left: ORB-SLAM2, top-right: RTAB-Map, middle-left: RGBDSLAMv2, middle-right: ManhattanSLAM, bottom: RESLAM.

4.2.3. Parking dataset

Taking a look at the statistics of error metrics for the parking dataset, shown in Table. 13, ORB-SLAM2 yet again performs quite remarkable, but with ManhattanSLAM not too far behind. RGBDSLAMv2 unfortunately struggled yet again with a dataset in an outdoor environment obtaining a max error of approximately 28.9 meters, which is relatively high when compared to the performance of the other SLAM systems, and the remaining statistics for the error metrics are not quite impressive either. RTAB-Map and RESLAM did not perform exceptional, but not necessarily terrible either, even though expectations for RESLAM was a bit higher after looking at the performance for the orchard dataset.

When looking at the plots, shown in Fig. 42 and Fig. 43, one could easily notice that RGBDSLAMv2 performs the worst, which is also indicative from the error metrics talked about earlier. Furthermore, the trajectory estimate for RGBDSLAMv2 could suggest that it struggles with turns as the estimate becomes erratic during the turns. ORB-SLAM2 and ManhattanSLAM seem to follow the trajectory the closest of all the SLAM systems, and whilst RTAB-Map and RESLAM performs decent, they seem to also struggle with the issue of estimating the trajectory when the robot turns, although not to the same degree as RGBDSLAMv2.

For the reconstructed maps in the parking dataset, shown in Fig. 44, most of the SLAM systems output a quite descriptive map of the parking lot environment. With the expectation from the orchard dataset in mind, ORB-SLAM2 and RTAB-Map especially, created a quite satisfactory, dense reconstruction of the environment, while RESLAM had a more sparse reconstruction of the map, it was definitely not dissatisfactory. Unfortunately, yet again, RGBDSLAMv2 had issues with the reconstruction here as it was not very descriptive of the parking environment. We see similar characteristics for ManhattanSLAM, where it seems to focus quite extensively on buildings around as well as the parking lot itself.

The statistics for each reconstructed map is shown in table 14. The ORB-SLAM2 had a file size of 80.42 MB and contained approximately 1,969,400 points within a bounding box volume of 19,700 cubic meters, resulting in a density of 99.72 points per cubic meter. Comparatively, the RTAB-Map file, with a larger file size of 100.09 MB, held a notably higher number of points at 3,748,223 within a significantly smaller volume of 7,800 cubic meters, resulting in an increased density of 480.74 points per cubic meter. Furthermore, the RGBDSLAMv2 file stood out due to its exceptionally large file size and volume, containing 8,996,388 points within 819,000 cubic meters, which interestingly led to a much lower density of approximately 10.98 points per cubic meter. On the other end of the spectrum, ManhattanSLAM, with the smallest file at only 1.21 MB, had the lowest point count and density, with just 45,398 points in a volume of 8,510 cubic meters, translating to an extremely sparse density of 0.053 points per cubic meter.

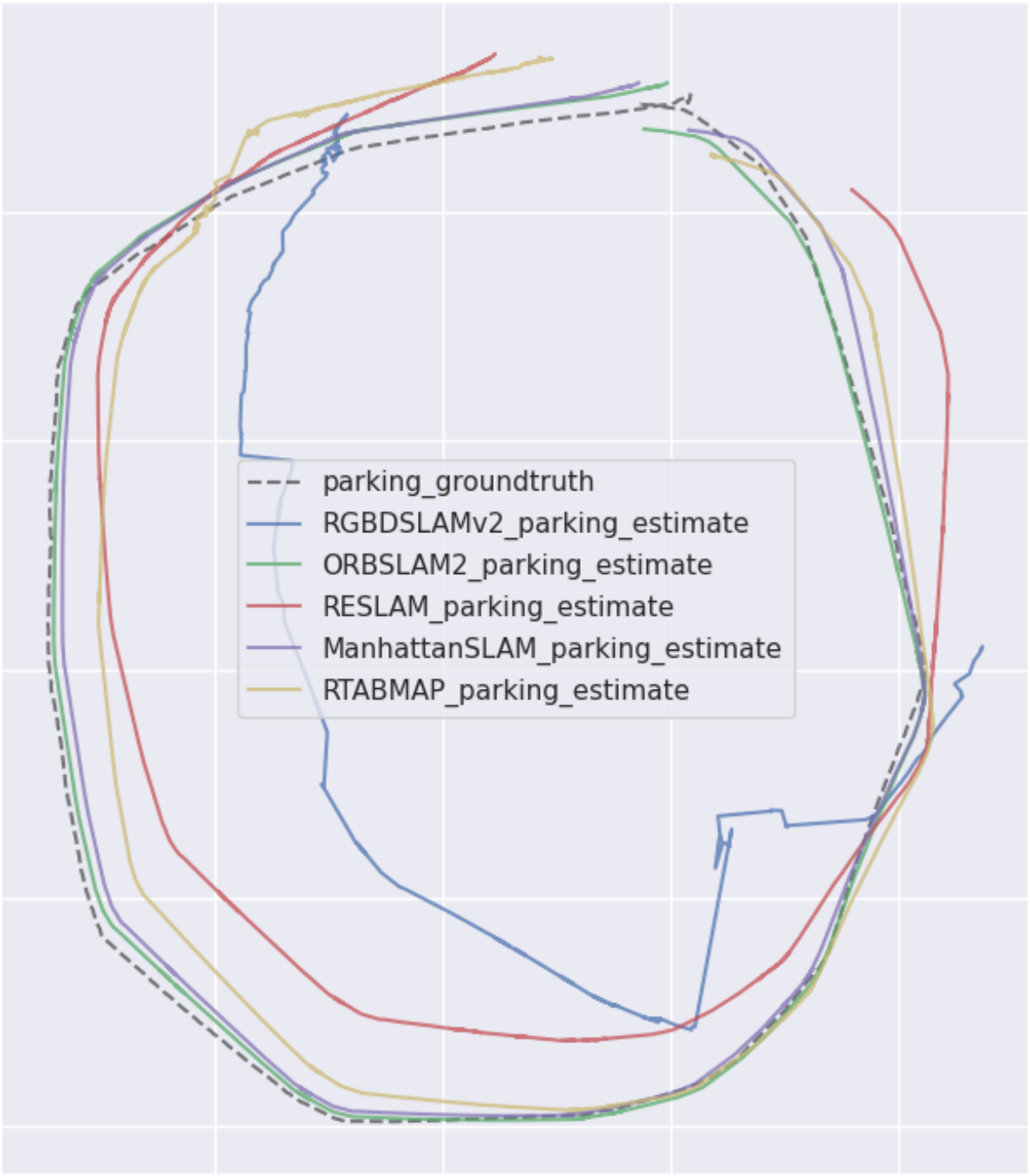


Fig. 42: XY plot of trajectory estimate from each SLAM system for the parking dataset with ground truth alignment using the Umeyama method [53].

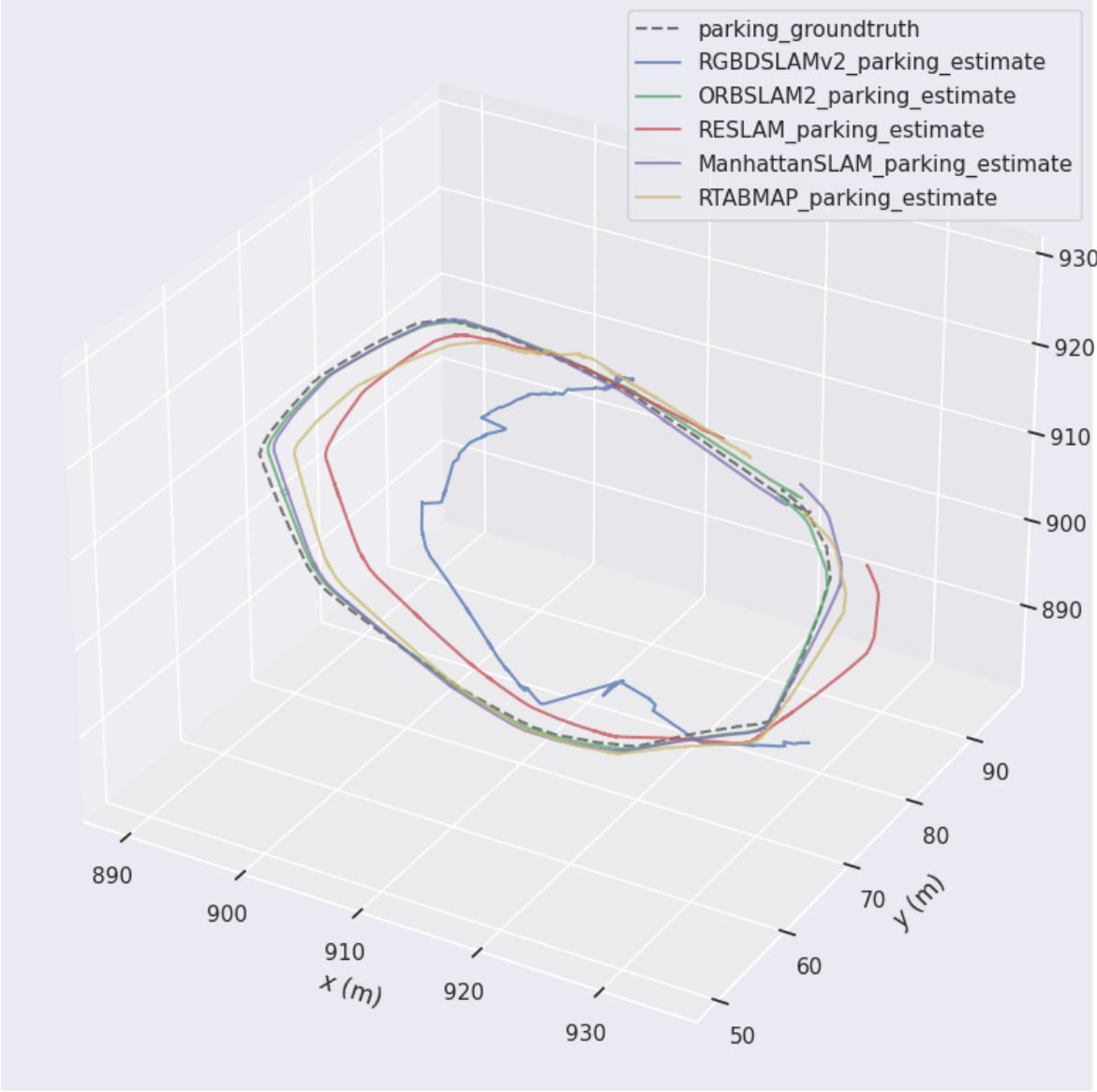


Fig. 43: XYZ plot of trajectory estimate from each SLAM system for the parking dataset with ground truth alignment using the Umeyama method [53].

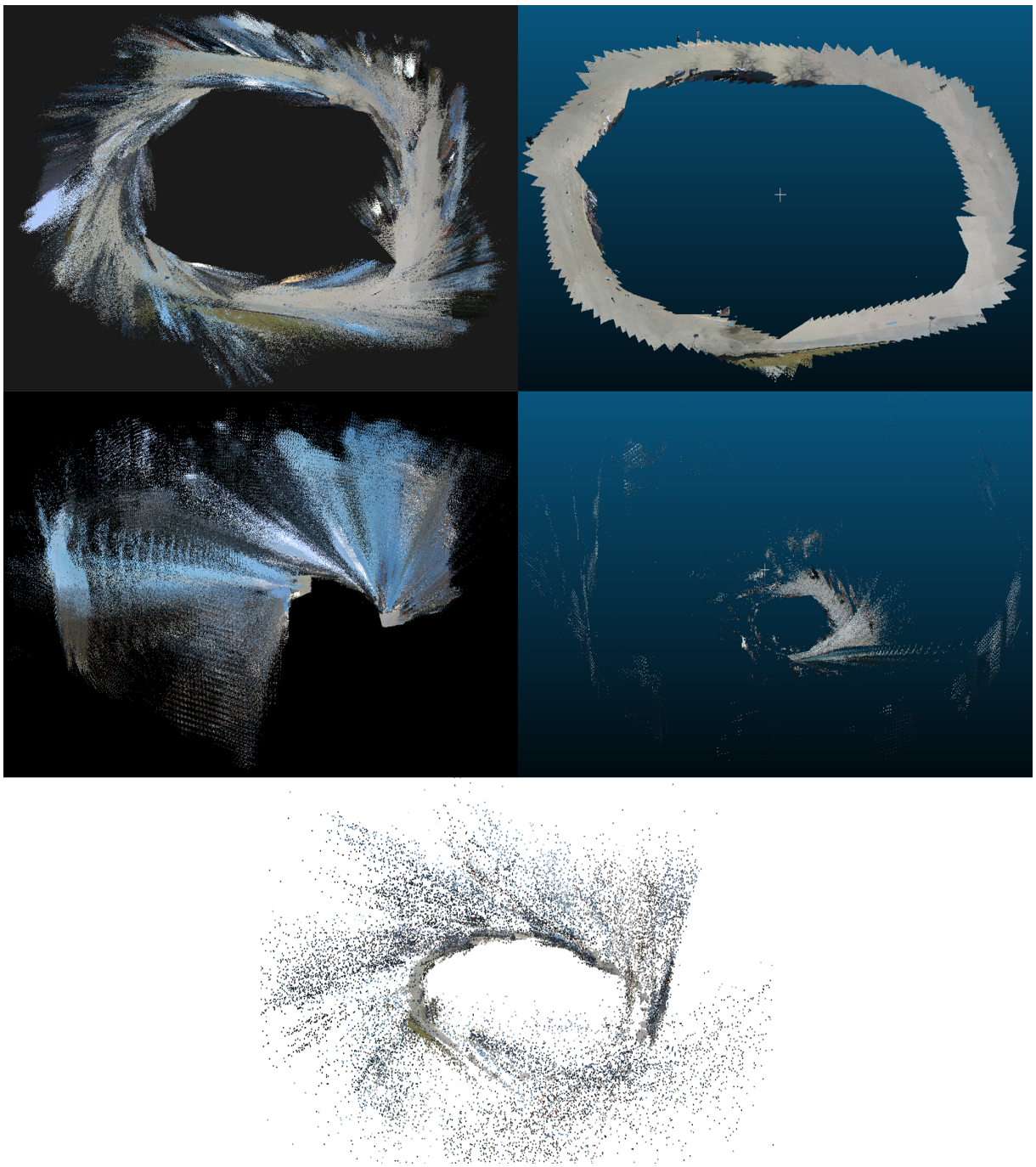


Fig. 44: Reconstructed maps for the parking dataset from each SLAM algorithm. Top-left: ORB-SLAM2, top-right: RTAB-Map, middle-left: RGBDSLAMv2, middle-right: ManhattanSLAM, bottom: RESLAM.

Algorithm	Max	Mean	Median	Min	RMSE	Std
ORB-SLAM2	**1.946445**	**0.758792**	**0.718813**	**0.163199**	**0.844977**	**0.371779**
RTAB-Map	7.157090	3.084035	2.955693	0.298086	3.563645	1.785579
RGBDSLAMv2	28.914690	11.422564	10.643130	1.878419	12.992568	6.191271
ManhattanSLAM	3.292745	1.212169	0.894768	0.177902	1.446434	0.789188
RESLAM	10.507336	5.070167	4.565861	0.652530	5.815401	2.848208

Table 13: Absolute positional error w.r.t. translation part (m) with Sim(3) Umeyama alignment [53] for the parking dataset. **All units are in meters.**

File Name	File Size (MB)	Number of Points	Volume (m^3)	Density (points/m^3)
ORB-SLAM2.pcd	80.42	1969400	1.97e+04	99.72
RTAB-Map.pcd	100.09	3748223	7.80e+03	480.74
RGBDSLAMv2.pcd	137.28	8996388	8.19e+05	10.98
ManhattanSLAM.pcd	1.21	45398	8.51e+03	0.053
RESLAM.pcd	N/A	N/A	N/A	N/A

Table 14: Statistics for each reconstructed map for the parking dataset. For each map in point cloud data (pcd) format, there is file size, total number of points, volume of the bounding box, and density of the points in terms of volume.

4.3. LIDAR

For the section of LIDAR based approaches the KITTI results will be presented first before presenting the data from the collected datasets.

4.3.1. KITTI

This section presents the performance of both KITTI odometry sequences. For each sequence, there will be tables of the Absolute Pose Error (APE) and plots of the trajectories. All KITTI plots share the legends found in Fig. 45. All measurements in the trajectory plots are in meters. And the origin is the start position for all plots.

4.3.2. KITTI 01

The evaluation metrics for the estimated trajectories are in Table 15 and show the numeric statistics. This table reveals significant performance variation between the approaches. LeGO-LOAM demonstrated the most consistent performance with an average error of 2.167 meters, followed by A-LOAM, F-LOAM and SC-A-LOAM which had average error of 8.274, 7.874 and 8.074 meters respectively. Similarly, the standard deviations show that LeGO-LOAM is the most consistent and 1.734 meters again followed by A-LOAM, F-LOAM and SC-A-LOAM at 2.657, 2.851 and 2.579. HDL Graph SLAM exhibited considerably

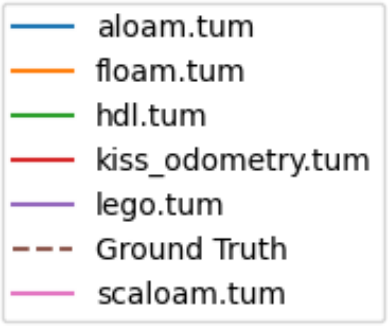


Fig. 45: Legend corresponding to the KITTI plots. Note that the ground truth shown as dotted lines while all the trajectories have full lines.

higher error rates with a maximum error at 650.768 meters, being 18 times higher than the second worst (KISS-ICP) and 98 times higher than LeGO-LOAM.

The presented 3D plots in Fig. 47 and in Fig. 47 visualize the estimated trajectories generated by each SLAM system. The ground truth is the dashed line and serves as a benchmark for the performance. LeGO-LOAMs trajectory matches best with the ground truth across all planes consistently to what table 15 presents. HDL Graph SLAM is the heaviest outlier, where even the origin of the generated trajectory does not match the origin of any other algorithm. This is visible in all planes, but is most significant in XZ and YZ plane, highlighting that HDL Graph SLAM struggles most with vertical accuracy. The other algorithms, A-LOAM, F-LOAM, SC-A-LOAM and KISS-ICP show good performance in the XZ plane, but they also show a decreased performance for vertical accuracy when inspecting XZ plane and YZ plane.

Fig. 48 shows a snapshot of the reconstructed maps generated for KITTI 01. Each sub-figure represents the individual map as generated by the respective algorithm. In terms of shape, LeGO-LOAM stands out as it generates a more curvy road compared to all the others. LeGO-LOAM, F-LOAM, A-LOAM and SC-A-LOAM gives color to the map based on the intensity of the scans while KISS-ICP and HDL graph SLAM generate a monochrome map. From Table 16 one finds that the file sizes of LeGO-LOAM, A-LOAM and SC-A-LOAM are quite similar, whilst LeGO-LOAM is boasting the smallest size at 58.7 MB. F-LOAM falls a bit behind this sitting at 82.9 MB, while KISS-ICP and HDL Graph SLAM are significantly larger than all the others at 116.9 MB and 209.7 MB, respectively. Considering the number of points the HDL Graph SLAM has the most points at 6347677, more than twice as many points as

Algorithm	Max	Mean	Median	Min	RMSE	Std
A-LOAM	14.55	8.274	7.579	3.479	8.69	2.657
F-LOAM	13.914	7.874	7.25	3.638	8.374	2.851
HDL-Graph-Slam	650.768	350.155	336.409	31.08	388.507	168.312
KISS-ICP	36.078	19.902	22.194	7.306	21.118	7.064
LeGO-LOAM	**6.651**	**2.617**	**2.43**	**0.583**	**3.139**	**1.734**
SC-A-LOAM	13.971	8.074	7.421	3.362	8.476	2.579

Table 15: Absolute positional error from each algorithm for the first sequence from the kitti dataset. **All units are in meters**

LeGO-LOAM, A-LOAM, SC-A-LOAM and F-LOAM and 1.68 times the number of points as KISS-ICP. Although the HDL Graph SLAM holds the highest number of points, it is also the smallest in volume at $3.3 \times 10^7\,\mathrm{m}^3$ a magnitude lower than all the other algorithms. Resulting in a density of $0.19\,\mathrm{points}/\mathrm{m}^3$.

File Name	File Size (MB)	Number of Points	Volume (m^3)	Density (points/m^3)
kiss.pcd	116.92000	3770678	3.16e+08	0.01194
lego.pcd	58.68000	1652093	1.46e+08	0.01135
hdl.pcd	209.66000	6347677	3.33e+07	0.19076
floam.pcd	82.92000	2119120	6.84e+08	0.00310
aloam.pcd	60.43000	1492762	6.21e+08	0.00240
scaloam.pcd	63.30000	1563504	6.22e+08	0.00251

Table 16: Statistics for each reconstructed map for KITTI 01 sequence. For each map in point cloud data (pcd) format, there is file size, total number of points, volume of the bounding box, and density of the points in terms of volume.

Table 17 represents the absolute positional error metrics for each of the different algorithms being applied to the KITTI 05 sequence and offers an insight on the performance. KISS-ICP offers a the most stable performance by having the smallest spread of error values, ranging from 0.381 to 5.393 meters. This is again reflected by KISS-ICP boasting the lowest mean error at 1.241 meters and the lowest standard deviation at 0.703 meters. This implies that the most accurate algorithms in this sequence is KISS-ICP. A-LOAM, and SC-A-LOAM also has excellent performance where the mean errors are 1.759 and 1.697 meters. Along with KISS-ICP, they are the only ones that keep the maximum error under 10 meters. LeGO-LOAM has a slightly higher mean of 2.775 meters and also has the largest recorded error at 10.363 meters. HDL Graph SLAM and F-LOAM fall in the middle range with mean errors of 2.519 and 2.387. HDL Graph SLAM presents the lowest minimum error of the algorithms, indicating that it has high accuracy under certain conditions. KISS-ICP has the most accurate performance for this sequence with A-LOAM and SC-A-LOAM also displaying good performance.

The following figures illustrate the estimated trajectories for the KITTI 05 sequence. The sub-figures in

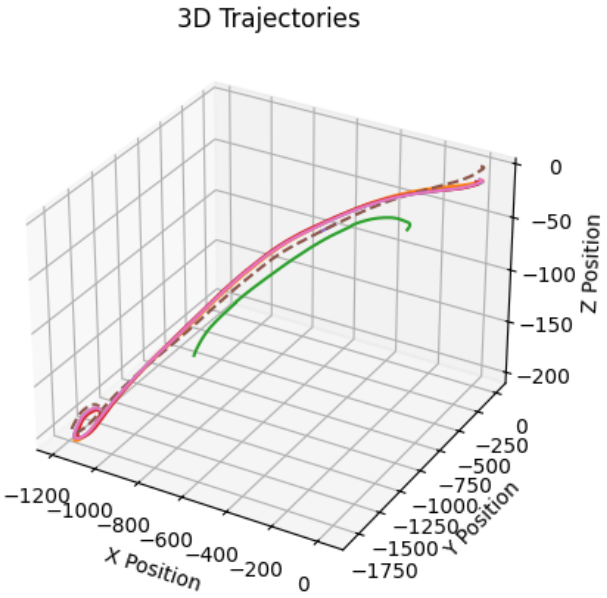


Fig. 46: XYZ plot of trajectory estimate from each SLAM system for the first seuqnce from Kitti with ground truth alignment using the Umeyama method [53].

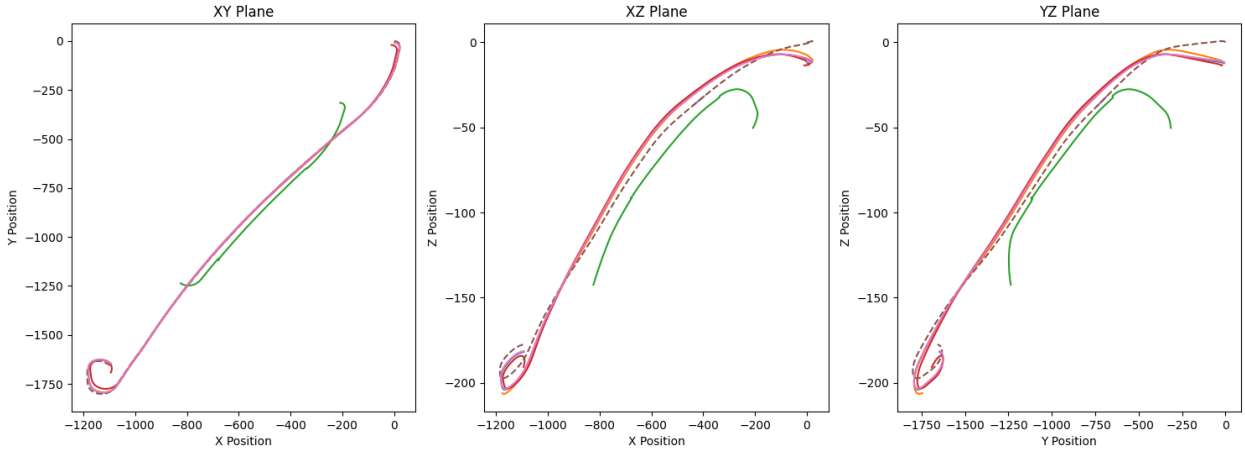
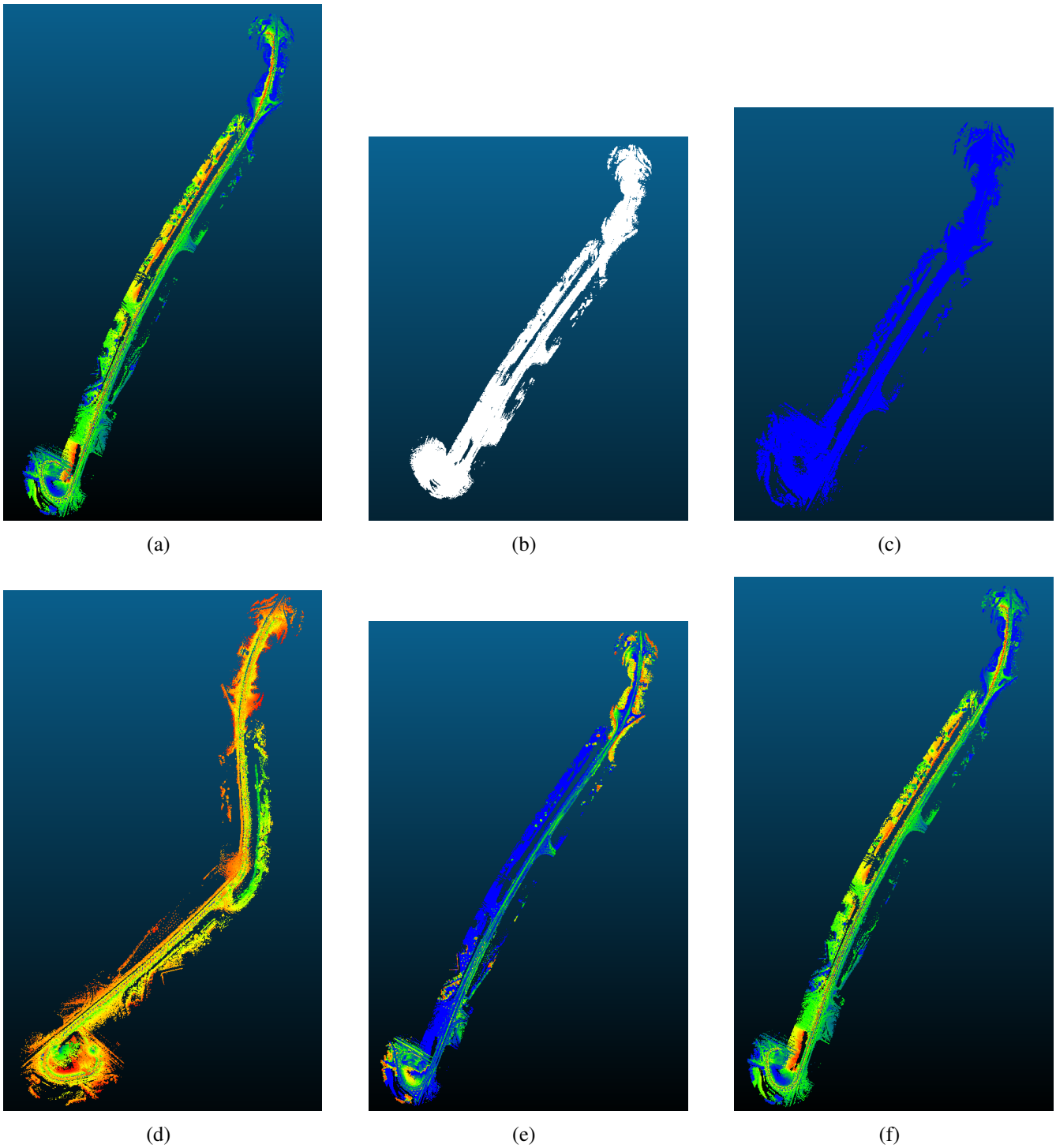


Fig. 47: XY, XZ and YZ plot of trajectory estimate from each SLAM system for the first sequence from Kitti with ground truth alignment using the Umeyama method [53].



(a) (b) (c)

(d) (e) (f)

Fig. 48: Reconstructed map for KITTI sequence 01. From top left: (a) SC-A-LOAM, (b) Kiss ICP, (c) HDL Graph SLAM, (d) LeGO-LOAM, (e) F-LOAM, (f) A-LOAM.

Algorithm	Max	Mean	Median	Min	RMSE	Std
A-LOAM	7.481	1.769	1.35	0.546	2.201	1.311
F-LOAM	8.44	2.519	1.998	1.034	2.927	1.49
HDL-Graph-Slam	6.639	2.387	2.427	**0.342**	2.646	1.142
KISS-ICP	**5.393**	**1.241**	**1.189**	0.381	**1.426**	**0.703**
LeGO-LOAM	10.363	2.775	2.476	0.668	3.177	1.545
SC-A-LOAM	7.202	1.697	1.311	0.565	2.102	1.24

Table 17: Absolute positional error from each algorithm for the fifth sequence from the KITTI dataset. **All units are in meters**.

Fig. 50 and Fig. 49 show the XY, XZ, YZ and the three dimensional consolidation of these images. These visualizations allows the performance to be qualitatively assessed and compare the performance in the individual dimensions. The 3D plot does not show any extreme outliers as all of the algorithms struggle to match the ground truth. When inspecting the 2D plots it is clear that all algorithms are capable of estimating the trajectory with respect to X and Y as in the XY plot the estimations and the ground truth align very closely. The XZ and YZ plot are much more noisy and indicates that the algorithms struggle more to estimate accurate altitude.

The reconstructed maps in Fig. 51 shows a top down view of the reconstructed maps. A-LOAM and SC-A-LOAM share the color pallet for the points while F-LOAM and LeGO-LOAM display more yellow and blue respectively. KISS-ICP and HDL Graph SLAM do not offer any information about the intensity of the points and only generates a monochrome representation. Taking a closer look at the intersection in the middle of each sub-figure in Fig. 52 they appear very similar and both of the algorithms generate two layers for the road while it is supposed to be one layer.

When inspecting table 18 we can get statistical information about the reconstructed maps. The file sizes are more spread out, the largest file being hdl.pcd at 240 MB and the smallest being lego.pcd at 30 MB. F-LOAM, A-LOAM and SC-A-LOAM have a similar size, ranging between 48 and 66 MB while kiss.pcd is 103 MB. The largest file is also the file with the highest number of points, while the smallest file has the lowest number of points. For the volume, aloam.pcd and scaloam.pcd have almost identical volume at $9.93 \times 10^6\,\mathrm{m}^3$ and $9.96\,\mathrm{m}^3$. lego.pcd has the lowest volume at $8.96 \times 10^6\,\mathrm{m}^3$ and the largest being floam.pcd $1.54 \times 10^7\,\mathrm{m}^3$. hdl.pcd has the highest density, having $0.67\,\mathrm{points/m}^3$ which is twice as dense as the runner up kiss.pcd at $0.247\,\mathrm{points/m}^3$. The final four files have a similar density, being around $0.67\,\mathrm{points/m}^3$.

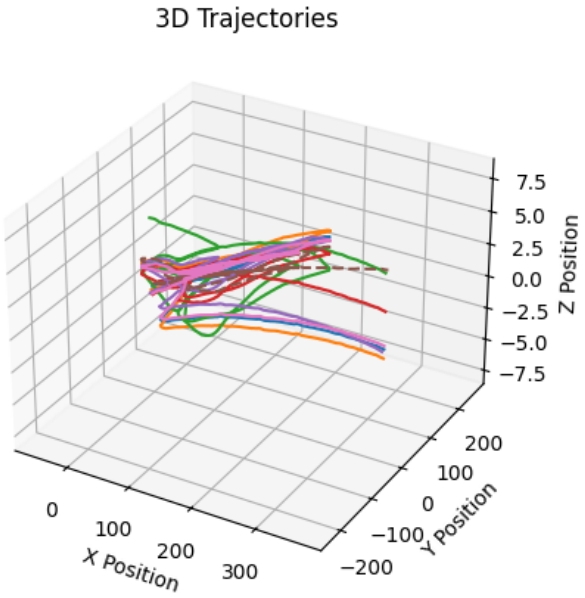


Fig. 49: XYZ plot of trajectory estimate from each SLAM system for the fifth sequence from KITTI with ground truth alignment using the Umeyama method [53].

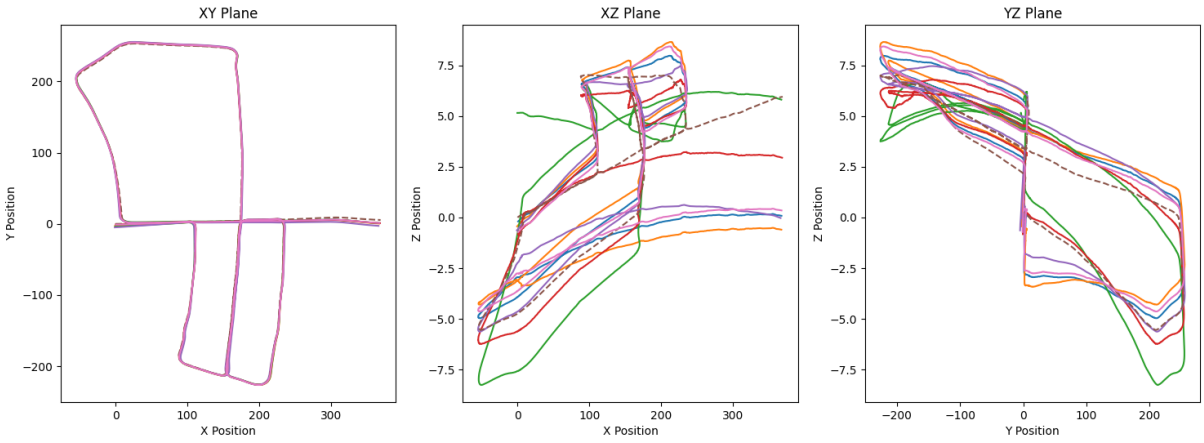
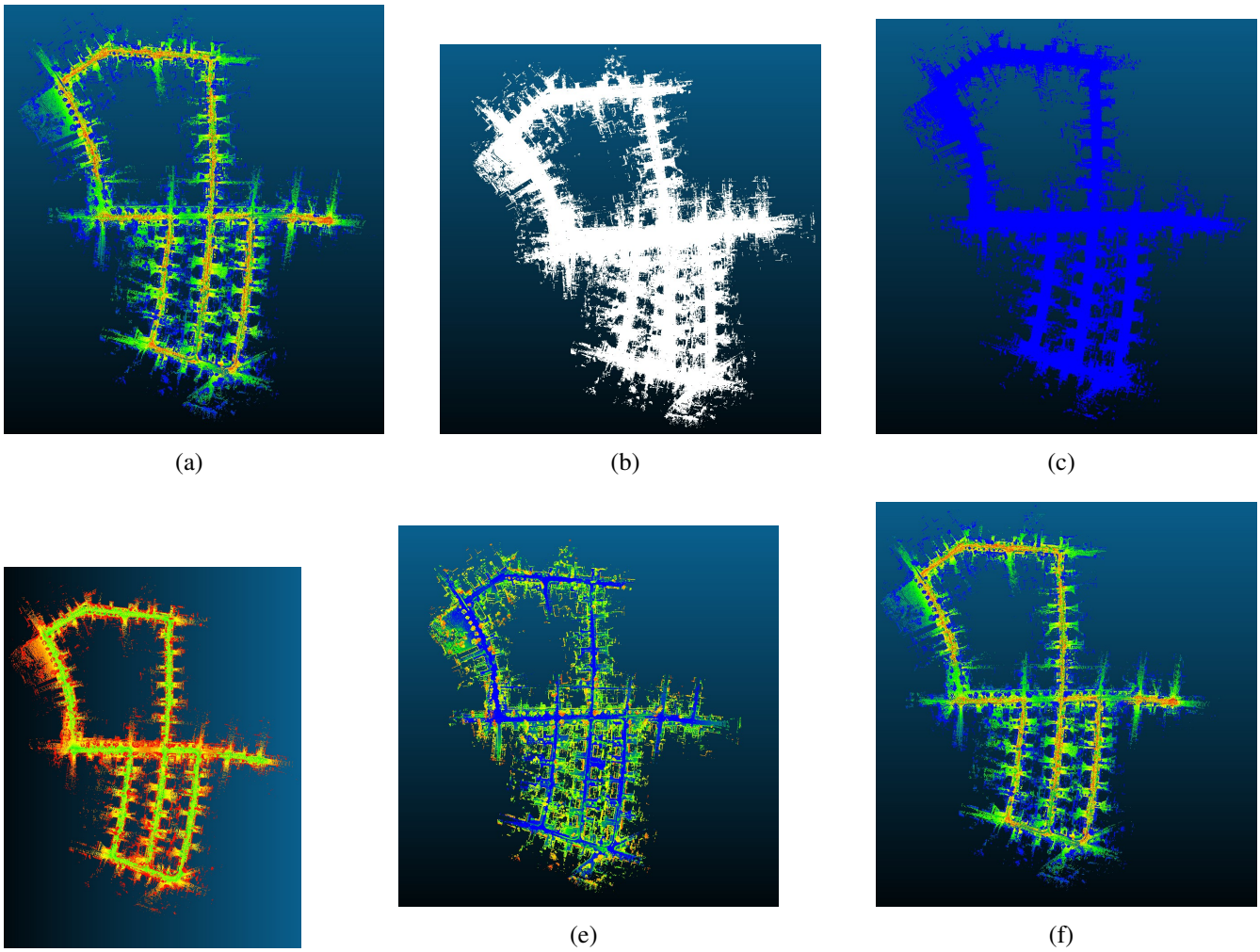


Fig. 50: XY, XZ and YZ plot of trajectory estimate from each SLAM system for the fifth sequence from KITTI with ground truth alignment using the Umeyama method [53].

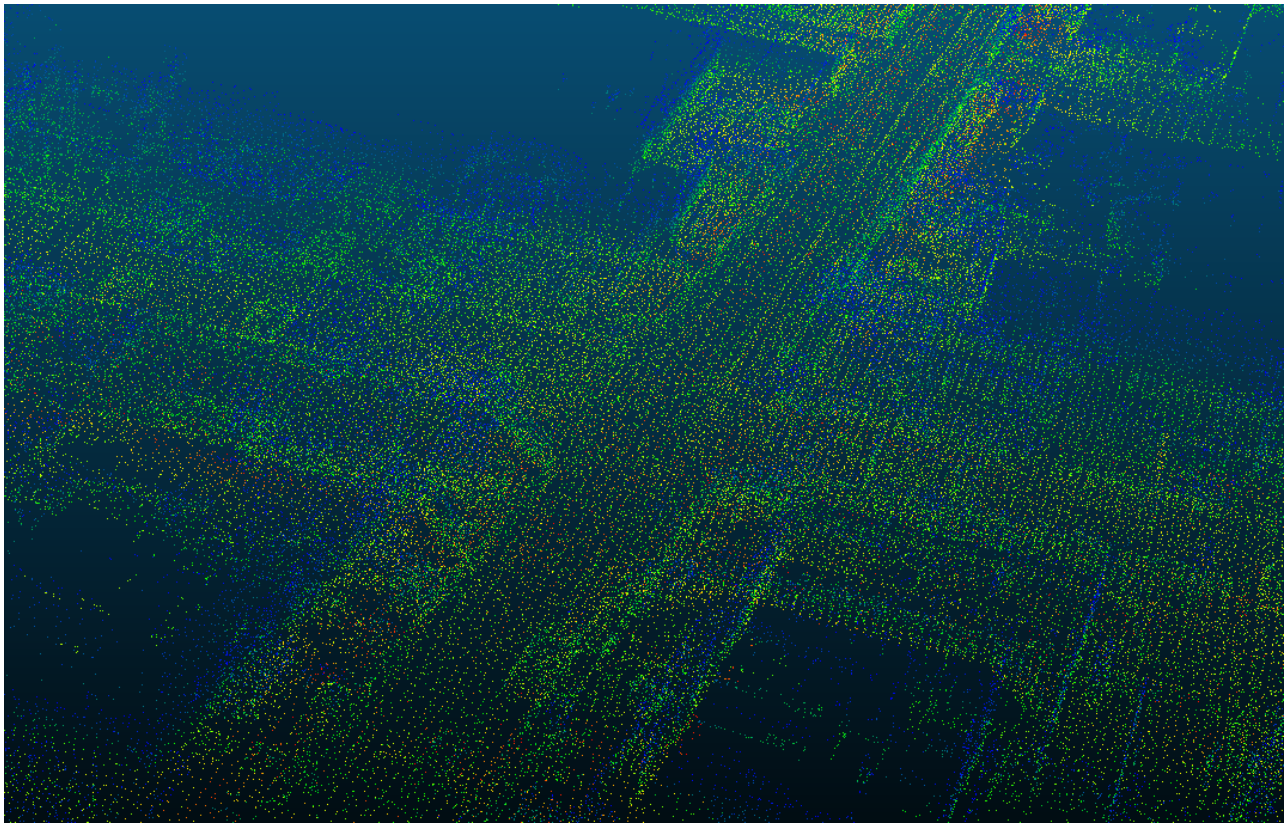
File Name	File Size (MB)	Number of Points	Volume (m^3)	Density (points/m^3)
kiss.pcd	103.17000	3484703	1.41e+07	0.24745
lego.pcd	30.66000	822944	8.96e+06	0.09183
hdl.pcd	240.52000	7617550	1.13e+07	0.67637
floam.pcd	65.84000	1642879	1.54e+07	0.10698
aloam.pcd	48.72000	1247896	9.93e+06	0.12564
scaloam.pcd	53.17000	1358872	9.96e+06	0.13637

Table 18: Statistics for each reconstructed map for KITTI 05 sequence. For each map in point cloud data (pcd) format, there is file size, total number of points, volume of the bounding box, and density of the points in terms of volume.

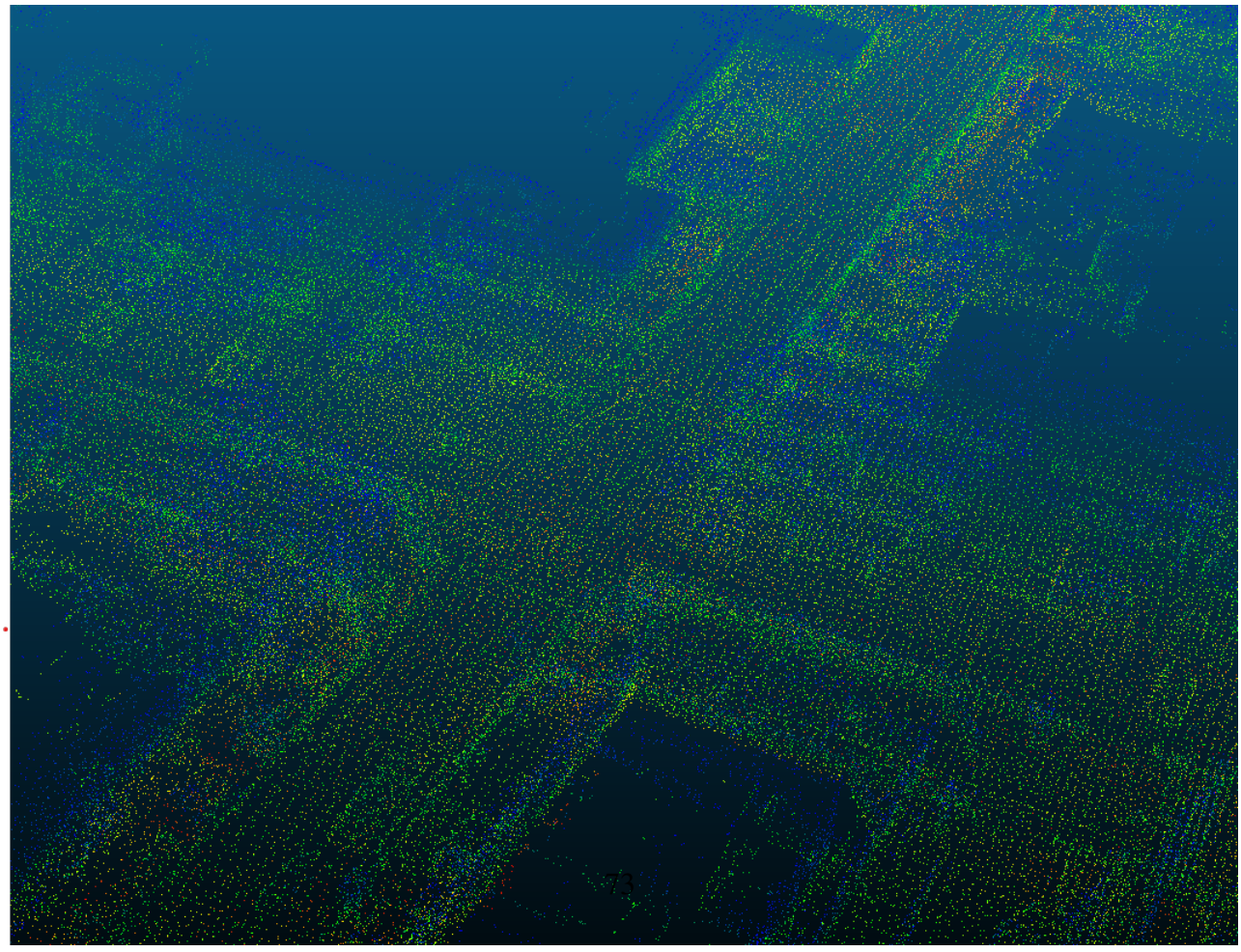


(a) (b) (c) (d) (e) (f)

Fig. 51: Reconstructed map for KITTI sequence 01. From top left: (a) SC-A-LOAM, (b) Kiss ICP, (c) HDL Graph SLAM, (d) LeGO-LOAM, (e) F-LOAM, (f) A-LOAM.



(a)



(b)

Fig. 52: Section of A-LOAM (a) and SC-A-LOAM (b) zoomed in to a intersection of the roads.

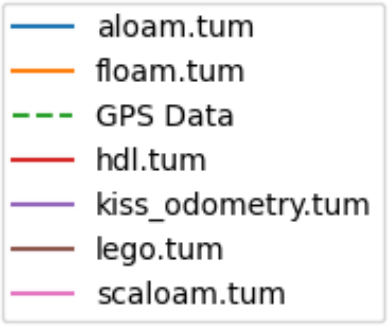


Fig. 53: Legend corresponding to the collected datasets and their corresponding plots. Note that the ground truth shown as dotted lines while all the trajectories have full lines.

4.3.3. Orchard dataset

Both of the self collected datasets share the legend which is in Fig. 53.

For the orchard dataset the absolute positional error can be observed in Table 19. All algorithms offer a very similar performance where the deviations between each algorithm is small. The maximum error varies from 1.437 meters to 1.673 meters with SC-A-LOAM and LeGO-LOAM respectively. SC-A-LOAM and A-LOAM has almost identical performances within all of the categories, only separated by 0.02 meters in minimum error which is the largest gap between them. HDL Graph SLAM outperformed the others in mean error, median error, RMSE and std which indicates that this is the best performing overall.

In Fig. 54 the 3D plot show how the trajectories are estimated from the orchard dataset, with trajectory alignment. The green dotted line represents the ground truth and when looking at the origin and endpoint for this line one can observe how they do not align, while in reality the robot started and ended at approximately same altitude. This is also reflected in 55 where the 2D plot for XY, XZ and YZ respectively are plotted. While the XY plane plot shows precise alignment of the algorithms and the ground truth, the XZ and YZ plane deviates more from the ground truth, but they all deviate in the same way which indicates that the algorithms produce a better trajectory than the RTK-GNSS system.

Figures 41 show how A-LOAM, F-LOAM, LeGO-LOAM and SC-A-LOAM maps dynamic object to the global map as the lines in between the rows of show how the data collectors moved along the robot to capture the data. The angle for HDL Graph SLAM and KISS-ICP is different and aims to make the

Algorithm	Max	Mean	Median	Min	RMSE	Std
A-LOAM	1.442	0.875	0.875	0.362	0.896	0.195
F-LOAM	1.454	0.883	0.879	0.389	0.904	0.192
HDL-Graph-Slam	1.545	**0.733**	**0.723**	0.36	**0.754**	**0.175**
KISS-ICP	1.495	0.855	0.849	**0.289**	0.878	0.202
LeGO-LOAM	1.673	1.098	1.061	0.716	1.12	0.22
SC-A-LOAM	**1.437**	0.871	0.868	0.382	0.891	0.19

Table 19: Absolute positional error from each algorithm for the dataset collected from an orchard at NMBU. **All units are in meters**.

resulting figure more readable as these are monochrome which makes the reconstructed map less readable. A closer inspection of KISS-ICP shows that there are stray points in the cloud, indicating a lack of outlier removal before mapping. When inspecting the statistical metrics as found in Table 20 the file sizes for these maps are between 1.6 and 5.4 MB by lego.pcd and kiss.pcd, respectively. Lego.pcd is an outlier on the lower boundary, as floam.pcd, aloam.pcd and scaloam.pcd are around 3 MB while hdl.pcd is 5 MB. The file size and number of points seem to correlate. Where more points correlate with larger size, kiss.pcd is both the largest and has the most points at 186750 points and lego.pcd has the smallest file and the fewest points at 43376 points. Where the maps deviate the most is when considering volume of the bounding box, this ranges from $7.35 \times 10^4\,\mathrm{m}^3$ to $1.81 \times 10^7\,\mathrm{m}^3$. The map with lowest volume is hdl.pcd while the highest is aloam.pcd. Hdl.pcd has both the highest number of points and the smallest volume that leads to the highest density $2.29\,\mathrm{points/m}^3$ lego.pcd had the fewest points and close to the largest volume leading to the lowest density of $0.002\,17\,\mathrm{points/m}^3$.

File Name	File Size (MB)	Number of Points	Volume (m^3)	Density (points/m^3)
kiss.pcd	3.61000	120482	5.84e+06	0.02062
lego.pcd	1.25000	32729	1.51e+07	0.00217
hdl.pcd	5.33000	168450	7.35e+04	2.29202
floam.pcd	2.08000	51678	6.62e+05	0.07811
aloam.pcd	2.40000	60660	1.80e+07	0.00337
scaloam.pcd	2.43000	61425	1.71e+07	0.00359

Table 20: Statistics for each reconstructed map for the orchard dataset. For each map in point cloud data (pcd) format, there is file size, total number of points, volume of the bounding box, and density of the points in terms of volume.

4.3.4. Parking dataset

Table 21 shows the quantitative results of the absolute positional error metric on the parking dataset collected at NMBU. These results highlight that LeGO-LOAM produces the most precise trajectory courtesy

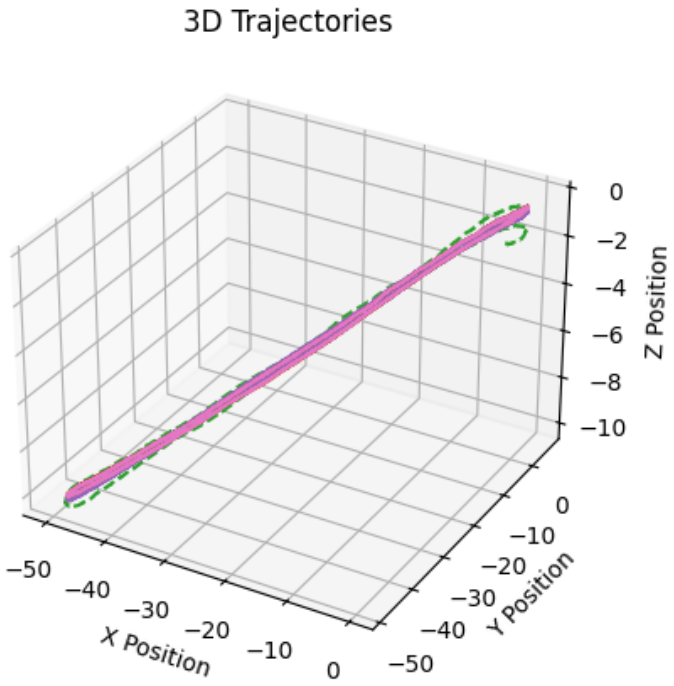


Fig. 54: XYZ plot of trajectory estimate from each SLAM system for the orchard dataset with ground truth alignment using the Umeyama method [53]. The green dotted lines is the ground truth.

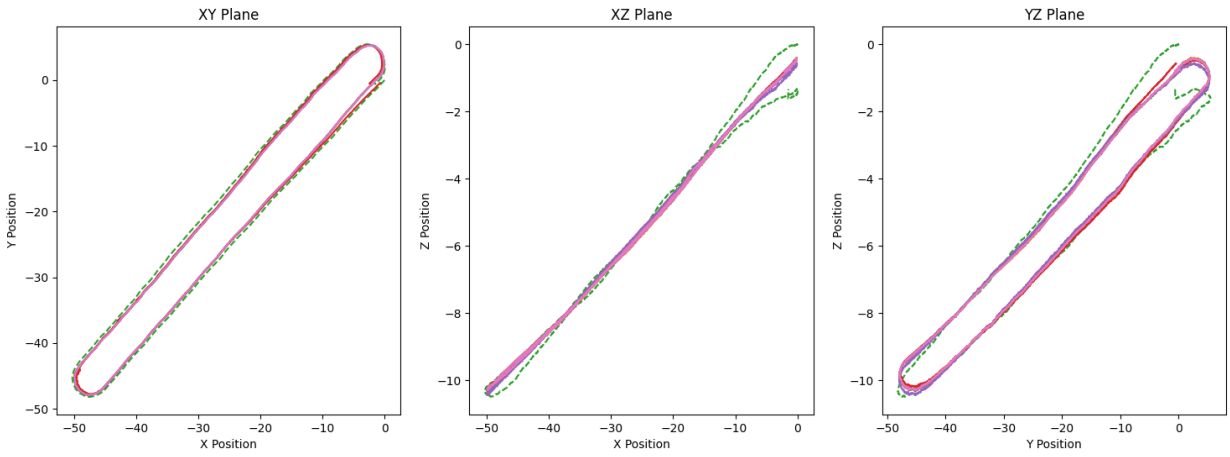
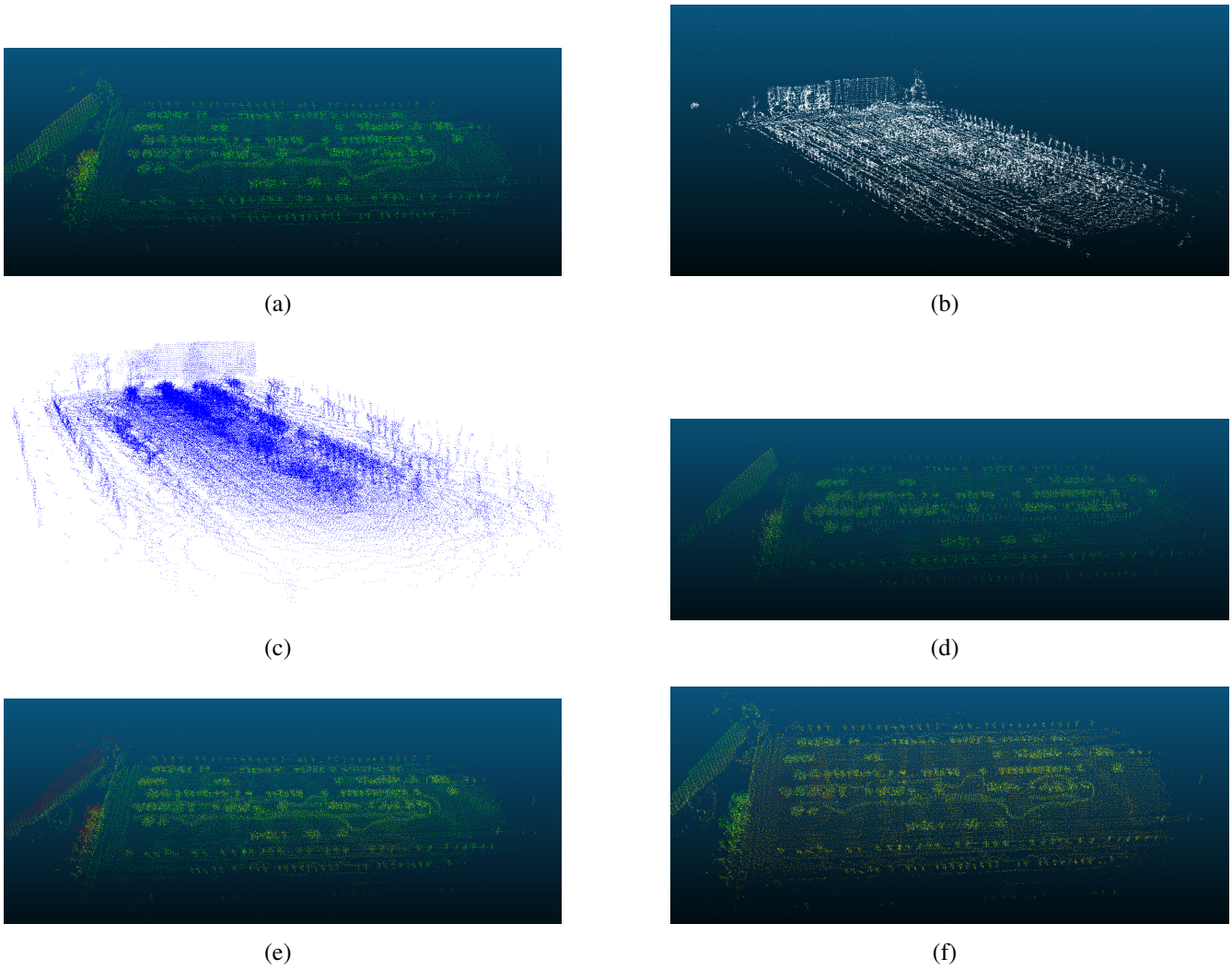


Fig. 55: XY, XZ and YZ plot of trajectory estimate from each SLAM system for the orchard dataset with ground truth alignment using the Umeyama method [53]. Legend is located in : Fig. 53.



(a) (b) (c) (d) (e) (f)

Fig. 56: Reconstructed map for orchard dataset. From top left: (a) SC-A-LOAM, (b) Kiss ICP, (c) HDL Graph SLAM, (d) LeGO-LOAM, (e) F-LOAM, (f) A-LOAM.

of having the lowest value in five of the six categories. Especially the RMSE score at 0.694 meters and the standard deviation at 0.25 meters show that the estimated trajectory is both consistent and accurate for this environment. In comparison, HDL Graph SLAM has competitive mean error, at 0.909 meters and the lowest minimum error of 0.254 meters suggesting periods of high accuracy. The maximum error on the other hand, is higher than LeGO-LOAM. The rest of the algorithms, A-LOAM, F-LOAM, KISS-ICP and SC-A-LOAM, show more variable performances. A-LOAM and SC-A-LOAM both boast a mean error under 1 meter, at 0.988 and 0.974 meters respectively. These two also has a almost identical standard deviation only separated by 0.001 meters. Overall the results show that the performances are quite competitive with LeGO-LOAM having the best overall performance for this dataset.

Fig. 57 offers a 3D visualization of the trajectory estimates for the parking dataset, aligned with the ground truth. All the algorithms seem to be fairly aligned, while the ground truth seems to be the outlier. This is further supported by Fig. 58 which shows the 2D plot for the trajectory estimates. The XY plane shows that the estimation algorithms slightly deviates from the ground truth, but that all of them deviate the same way. In the XZ and YZ planes the deviation between the algorithms and ground truth is more significant. These plots indicate that altitude is the least stable dimension both for the ground truth and for the algorithms.

The reconstructed map in Fig. 44 shows each of the reconstructed maps for the parking dataset. KISS-ICP seems to be the worst at removing outliers as there are some points far from any object. A-LOAM and SC-A-LOAM share an almost identical color pallete to represent the intensity of the scans. LeGO-LOAM and F-LOAM uses a different color palette while still being similar in style. KISS-ICP and HDL Graph SLAM have monochrome representation. Taking a closer look at the statistics of the files as found in Table 20 the file size varies from 1.6 to 5.4 MB with lego.pcd being the smallest file and kiss.pcd being the largest file. Aloam.pcd and scaloam.pcd have identical files size at 3.44 MB, floam.pcd is slightly smaller at 2.73 MB and hdl.pcd is larger at 5.01 MB. As for the total number of points, kiss.pcd has the most points having 186750 while lego.pcd has the lowest amount of 43376. aloam.pcd and scaloam.pcd are separated by 50 points. Hdl.pcd has the smallest volume of $2.26 \times 10^5\,\mathrm{m}^3$, while scaloam.pcd has the largest volume $2.97 \times 10^7\,\mathrm{m}^3$.

Algorithm	Max	Mean	Median	Min	RMSE	Std
A-LOAM	2.388	0.988	0.942	0.372	1.072	0.416
F-LOAM	2.524	1.055	0.991	0.385	1.143	0.44
HDL-Graph-Slam	2.13	0.909	0.857	**0.254**	0.975	0.351
KISS-ICP	2.49	1.038	0.983	0.395	1.125	0.434
LeGO-LOAM	**1.496**	**0.647**	**0.604**	0.263	**0.694**	**0.25**
SC-A-LOAM	2.375	0.974	0.925	0.359	1.059	0.417

Table 21: Absolute positional error from each algorithm for the dataset collected from a parking lot at NMBU. **All units are in meters**.

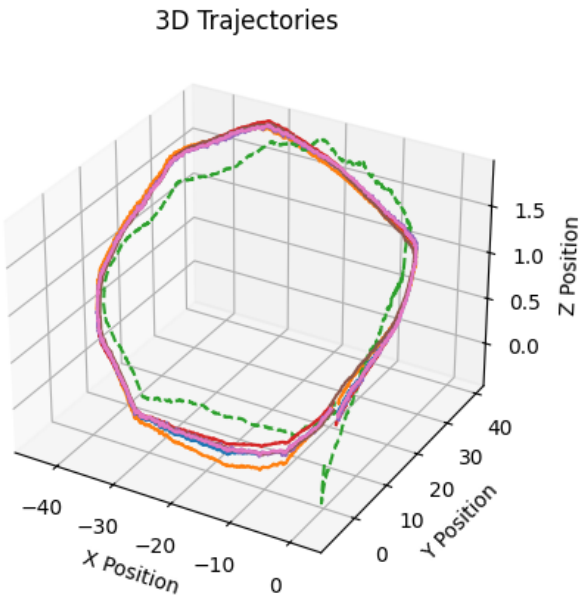


Fig. 57: XYZ plot of trajectory estimate from each SLAM system for the parking dataset with ground truth alignment using the Umeyama method [53]. Legend is located in : Fig. 53.

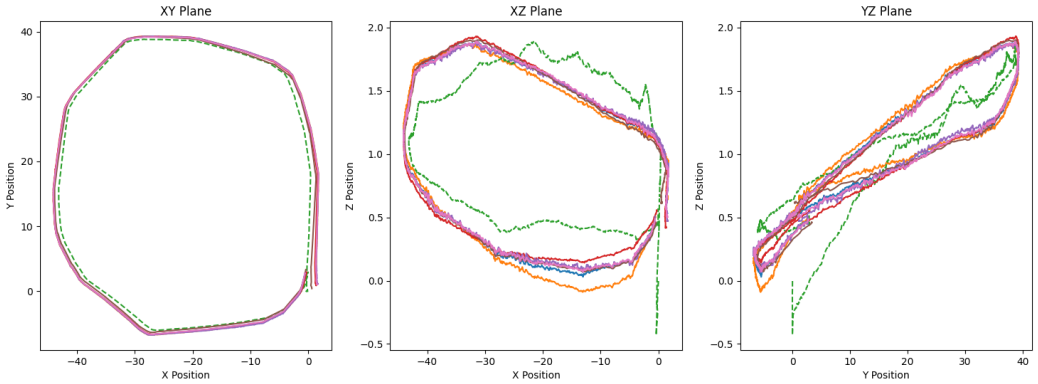
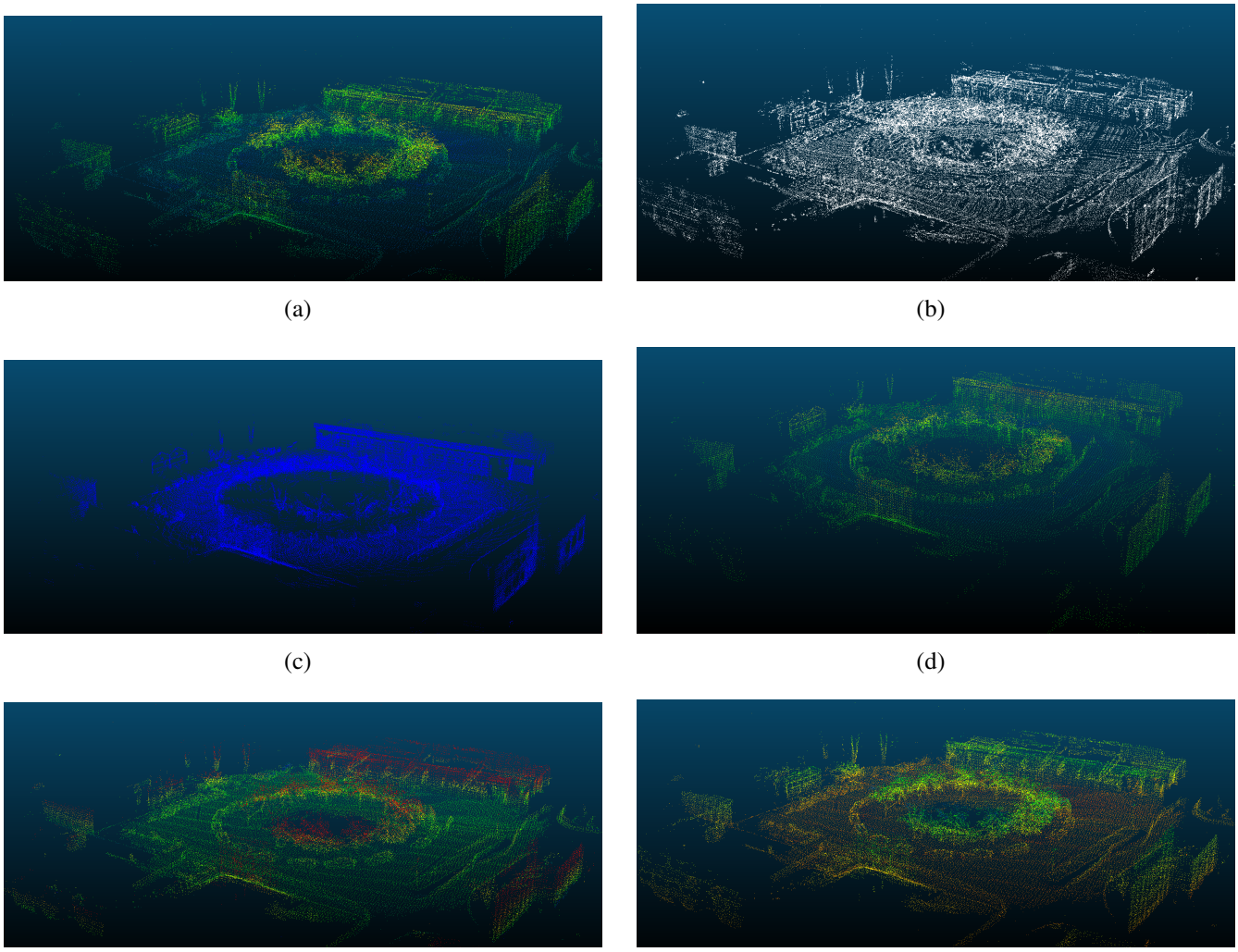


Fig. 58: XY, XZ and YZ plot of trajectory estimate from each SLAM system for the parking dataset with ground truth alignment using the Umeyama method [53].

File Name	File Size (MB)	Number of Points	Volume (m^3)	Density (points/m^3)
kiss.pcd	5.45000	186750	5.42e+06	0.03445
lego.pcd	1.62000	43376	1.21e+07	0.00359
hdl.pcd	5.01000	160877	2.26e+05	0.71312
floam.pcd	2.73000	71078	1.48e+06	0.04801
aloam.pcd	3.44000	88934	2.43e+07	0.00366
scaloam.pcd	3.44000	88884	2.97e+07	0.00300

Table 22: Statistics for each reconstructed map for the parking sequence. For each map in point cloud data (pcd) format, there is file size, total number of points, volume of the bounding box, and density of the points in terms of volume.



(a) (b) (c) (d) (e) (f)

Fig. 59: Reconstructed map for parking lot dataset. From top left: (a) SC-A-LOAM, (b) Kiss ICP, (c) HDL Graph SLAM, (d) LeGO-LOAM, (e) F-LOAM, (f) A-LOAM.

5. DISCUSSION

5.1. Ground truth accuracy

In the parking dataset and the orchard dataset, an RTK-GNSS receiver from ublox was used to estimate a ground truth, and under optimal conditions could provide centimeter-accuracy, as shown in Fig. 25 and 26. From these figures one can note that for the orchard dataset the accuracy is consistent, but for the parking dataset the data is less stable. If the error between the output of each SLAM system and the ground truth is *less* than the error estimate from the RTK-GNSS system, then these SLAM systems can be considered as accurate as RTK-GNSS in estimating the trajectory. This is especially important to consider for the altitude, as the covariance plots show that the estimated errors are larger in this dimension.

5.2. Camera

This section discusses the implication of the results of the camera based SLAM systems in both of the datasets. The discussion will display the contrasts of outdoor versus indoor performance, underlined by the error metrics and the quality of reconstructed maps w.r.t number of points, volume and density. Analysis will include both within-dataset comparisons and between the outdoor dataset collected for this study and the indoor TUM-RGBD dataset. By addressing these aspects through quantitative and qualitative assessments, this section aims to identify trends and deviations in system performance under different environmental conditions.

5.2.1. Environmental impact on camera-based SLAM performance

The comparative analysis illustrates the significant impact of environmental complexity on the performance of SLAM systems. Indoor environments, as exemplified by the TUM RGB-D dataset, generally provide favorable conditions for SLAM operations. These environments typically includes consistent features and controlled lighting, as well as minimal occlusions. These environments are densely populated with distinct, high-contrast textures that aid in reliable feature detection and matching. For instance, the fr2_desk sequence demonstrated the strengths of each SLAM system, where all of them achieved exceptionally low error metrics and produced detailed, high-fidelity maps. This can be attributed to the SLAM system's reliance on rich textural information to maintain robust tracking and accurate map reconstructions.

In contrast, outdoor environments present a different array of challenges and complexities, as observed in the results for the orchard and parking datasets. These settings are characterized by variable lighting conditions, which can dramatically affect the visibility of features. For example, shadows and overexposure can obscure important landmarks or introduce noise into feature detections. The expansiveness of these environments also introduces challenges in maintaining persistent feature tracks across wide areas,

exacerbating issues with drift over long distances. The less structured and more dynamic nature of outdoor scenes further complicates the task, as moving objects and changing scene geometries can disrupt the continuity of feature observations.

The orchard dataset highlighted these challenges, where even the best-performing algorithms like ORB-SLAM2 and RTAB-Map exhibited increased error metrics compared to the indoor environments. Despite these challenges, RTAB-Map's dense mapping capabilities showcased its utility in environments requiring detailed environmental modeling. It maintained a higher density of points per cubic meter in its map reconstructions than other systems, suggesting that its approach to feature integration and map updating is well-suited to handling the complexities of outdoor navigation.

Moreover, the parking dataset underscored the variability in performance across different outdoor scenarios. Here, RGBDSLAMv2 faced significant difficulties, possibly due to how it handled vast open spaces with sparse features as well as its sensitivity to lighting variations, which led to the highest maximum error observed in the study. This contrasted sharply with its performance in more confined and well-defined indoor spaces, where it obtained the lowest error metrics for both the fr3_walking_xyz and fr3_sitting_static sequence. Thus, highlighting the importance of algorithmic adaptability to different environmental conditions.

These observations suggest that while current SLAM systems are capable of operating under a variety of conditions, there remains a substantial divide in performance between controlled indoor environments and unpredictable outdoor settings. Enhancements in more advanced feature extraction algorithms that can operate robustly in feature-poor regions, could help bridge this gap. Additionally, incorporating adaptive strategies that adjust the processing tactics based on the detected environment could further enhance the versatility and reliability of SLAM systems in diverse operational scenarios.

5.2.2. Quality of reconstructed maps

The quality of maps reconstructed by various SLAM systems can be deeply analyzed by examining the metrics w.r.t number of points, volume, and density. In controlled, indoor environments, as observed in the TUM RGB-D sequences, the systems generally produced maps with higher detail and density. This is primarily due to the consistent and structured nature of these environments, which are rich in distinct, static features that makes precise feature matching and robust tracking possible which are exemplified by the results in e.g. the fr2_desk sequence for all of the SLAM systems.

Conversely, in outdoor settings such as the orchard and parking datasets, the challenges imposed by expansive areas, variable lighting, and less structured scenes are evident in the reconstructed maps. The orchard dataset presented a diverse performance across systems. ORB-SLAM2 and RTAB-Map were notable for maintaining a reasonable balance between detail and density despite the challenging environment.

ORB-SLAM2 generated a map with approximately 2.4 million points in a volume of 11,600 cubic meters, achieving a density of 207.03 points per cubic meter. RTAB-Map, on the other hand, demonstrated its dense mapping capability by achieving an impressive density of 3,732.91 points per cubic meter within a much smaller volume of 2,820 cubic meters. This high density signifies RTAB-Map's proficiency in capturing the detailed environmental features, crucial for tasks requiring accurate and detailed environmental modeling.

In stark contrast, RGBDSLAMv2 and ManhattanSLAM struggled in these less controlled outdoor settings. In the orchard dataset, RGBDSLAMv2's map contained an enormous number of points—approximately 43.9 million—spread over a vast volume of 3.19 million cubic meters, resulting in a very low density of 13.77 points per cubic meter. This low density suggests inefficiency in point distribution and possible difficulties in handling expansive and feature-poor environments. ManhattanSLAM showed even more pronounced challenges, with a map density of only 0.35 points per cubic meter, highlighting its struggle with vast, unstructured environments. This suggests that the Manhattan World assumption that ManhattanSLAM is based on, does not perform well in an agricultural environment with few orthogonal and planar features.

The parking dataset further enhanced these disparities. Here, the challenges were magnified by the expansive open spaces typical of parking areas, which are often devoid of distinct, close-range features that SLAM systems rely on for accurate mapping. RGBDSLAMv2's performance in this dataset was notably poor, with a map containing about 8.99 million points but spread over a tremendous volume of 819,000 cubic meters, leading to a density of only 10.98 points per cubic meter. Such a low density in a complex outdoor environment suggests difficulties in maintaining feature tracking and map coherence over larger distances and in feature-sparse regions.

Yet again it is evident that environmental conditions play a significant role in influencing the performance of SLAM systems, not only in terms of trajectory estimates, but also in map reconstructions. Indoor environments, with their abundance of clear, static features, allow for more detailed and dense map reconstructions. In contrast, outdoor environments challenge these systems with their scale, complexity, and variable conditions, often resulting in lower fidelity and density in the reconstructed maps. Improving SLAM systems' robustness and adaptability to diverse and challenging outdoor environments remains a crucial area for future research and development.

5.3. LIDAR

In this section the variation in ground truth will be discussed and compared to the estimated position error for the LIDAR based algorithms. Then the LOAM based approaches of F-LOAM, A-LOAM, SC-A-LOAM and LeGO-LOAM will be compared within this subset. Lastly, the metrics for absolute positional

error and their implications will be explored before finishing with the density of the reconstructed maps.

5.3.1. Rolling Shutter in LIDAR

As the LIDAR used in this thesis produces scans at 10 Hz, higher velocities will get more smearing. At the highest velocity, 112 kilometers per hour in KITTI 01, the distance traveled in one second is 31 meters, meaning that between each rotation the platform will at least have moved 3 meters, making the scan matching vastly more challenging. Add the factor that not all scans are taken into consideration and the distance increases further. With this in mind, the camera has an advantage as the continuous capture allows for a reduced distance traveled between matching attempts. This further punishes algorithms that require more time for each calculation, as the distance for the scan matching will increase.

5.3.2. Variation in ground truth vs estimated position error

For all LIDAR datasets, the ground truth is calculated using some form of RTK-GNSS system, and for the KITTI dataset we cannot adjust this ground truth estimate in any way. Thus, we must assume that this ground truth is high precision. For our own dataset on the other hand, one has to consider the covariance of the estimation as found in Fig. 25 and Fig. 26. Firstly, in the orchard dataset, the covariance, especially in longitude and latitude, shows high accuracy. Comparing the results from Table 19 to the covariance plot in Fig. 25 we can observe that even the lowest mean, 0.733 meters, is larger than the $\sqrt{covariance_{max}} = 0.59$ which is the approximate maximum value of the least accurate dimension, altitude. That means that all algorithms have more variation than the variation explained by the inaccurate RTK-GNSS signal. For the covariance of Fig. 26 compared to Table 21, the highest value of the covariance is approximately $\sqrt{covariance_{max}} = 0.9$ while LeGO-LOAM has a mean of 0.604. This indicates that the entire variation in LeGO-LOAM might be due to the inaccurate ground truth estimates for this dataset, leading to a best-case scenario where LeGO-LOAM has equal performance to trajectory estimates as the RTK-GNSS approach. It is important to note that this compares the point in time where the variance in GNSS is the worst with the mean error of the entire dataset.

5.3.3. KITTI vs collected sequences

The four sequences can be classified as KITTI or collected. The KITTI sequences are about 10 times longer in path length compared to collected sequences, while still being relative similar in duration, meaning there are higher velocities for the KITTI dataset. Larger velocities translate to a larger distance traveled between scans; this is extra punishing for the methods where the computation is the slowest and infrequent. HDL Graph SLAM appears to be a victim of this as the performance in KITTI 01, as shown in Table 15 and the plot in Fig. 46 shows that HDL Graph SLAM misses the origin and never seems to recover. This case is further supported by the fact that HDL Graph SLAM displayed much more competitive results for

all other sequences as seen in tables 17, 19 and 21 as these sequences have a significantly lower average velocity than KITTI 01. Moreover, the higher velocities also affects the other algorithms as they all, except LeGO-LOAM, have their worst performance in KITTI 01. The previous case is also true here, where the dataset with the second highest velocity KITTI 05 shows much better performance for all of the remaining SLAM systems.

5.3.4. LOAM based approaches

For the LOAM based methods, A-LOAM, F-LOAM LeGO-LOAM and SC-A-LOAM, the results from KITTI 01 in Table 15 show LeGO-LOAM outperforming the rest by a significant margin. The other three methods perform very similar in this dataset. As KITTI 01 does not have any loops to close, SC-A-LOAM does not have the opportunity to utilize scan matching. In KITTI 05, where there are loop closures in the dataset, the SC-A-LOAM slightly outperformed the other LOAM methods. Fig. 52 shows how the SC-A-LOAM better recognizes the intersection. For both collected datasets, SC-A-LOAM outperformed F-LOAM and A-LOAM, with LeGO-LOAM being the worst performer in the orchard dataset and the best performer in the parking dataset. This inconsistency for LeGO-LOAM indicates that it is more environment-dependent, giving less reliable performance. One reason for this inconsistency could be that the pre-processing of the scan removes points in a way that hurts performance in some environments. Exemplified, if it is tuned to find the corners of flat surfaces it will suffer in the orchard as there is only one wall while all other sequences consists of more flat surfaces.

5.3.5. Reconstructed maps, file size, volume and density

When comparing the file metrics from the reconstructed maps as they are in tables 16, 18, 20 and 22, there is a deviation between the KITTI sequences and the collected sequences. Both file sizes and the number of points are larger in the KITTI sequences, and this might be due to the much longer paths and hence much larger area to map. The duration of the sequences contributes less to the file size, as the orchard dataset is 2.17 times longer than the parking dataset, while the average increase in file size between the sequences is 31.79%.

This is further supported when comparing the KITTI and collected sequences. Table 1 shows that KITTI path lengths are on average 2334 meters compared to 154 meters from Table 2. This equals to 15.15 times longer average path while the average file size increase is 31.6 times larger. Although the file size increases more than the path length, the KITTI duration is on average shorter than the both of the collected sequences. Considering that the map increases in three dimensions as the platform travels through the environment, it is consistent that an increasing path length would affect the file size more than the linear duration.

The highest density of points across all sequences is produced by HDL Graph SLAM, as it stores a higher

number of points than all other SLAM systems while keeping the volume of the bounding box small. This increase in number of points and density gives the method more points to match against, leading to heavier computing costs per scan matching. This increased cost is not necessarily only bad, as more complete information can lead to more accurate estimates. The LOAM based approaches apply more feature extraction and outlier remover such that the number of points in the final map has fewer points. Notably, LeGO-LOAM optimizes to reduce computational cost leading to the lowest average of points across the sequences.

5.4. LIDAR vs Camera

To discuss the differences in terms of performance and applicability between camera and LIDAR based systems this section will mainly focus on the results from the collected dataset as they were the only dataset that all of the SLAM systems were evaluated on. This section will therefore explore how adaptable each of the sensor-based SLAM systems are in the same environment, and provide theories as to *why* this is the case.

There are several nuanced factors that contribute to the different performance observed between each of the sensor-based SLAM systems. LIDAR technology, in contrast to cameras, relies on light detection and ranging to map physical spaces, meaning that it is inherently less susceptible to visual limitations that may affect camera-based systems. Example wise, in environments like orchards and parking lots that are characterized by complex geometries and variable lighting conditions, LIDAR systems excel by providing accurate spatial data, which is independent of ambient light.

In orchards, there are challenges due to structured, yet repetitive nature, and although this seems counterintuitive, this is especially true for cameras. LIDAR systems handle these scenarios better as they are more adept at capturing subtle differences between the rows of trees and the spacing between each of them. This is troublesome for camera images due to shadowing or overlapping branches, as can be seen from the results.

Some of the aforementioned capabilities of LIDAR systems comes to light in the parking lot dataset. Here, they benefit from being able to detect and map diverse structures, from building facades to cars, again, regardless of lighting conditions. This, unfortunately, yet again is an issue where cameras might struggle, where there is either too strong reflective light, or too little light with low visual contrast. In our case, as the recording was done during a very sunny day, the strong reflective light would be the culprit.

With all of this in mind, these can provide a basis as to why the each of the sensor-based SLAM systems perform the way they do, where all of the LIDAR based SLAM systems perform better than all of the camera based SLAM systems w.r.t RMSE and most of the other error metrics for both the orchard and parking dataset. Interestingly enough however, ORB-SLAM2 performs better than almost all LIDAR

based SLAM systems except for LeGO-LOAM on the parking dataset. There may be a multitude of reasons for this, where firstly, parking areas are typically more contained and enclosed, which can alleviate issues such as feature scarcity over large distances. The relatively static nature of the environment, also has a significant impact on ORB-SLAM2's performance, where compared to dynamic environments, static environments make it much easier to achieve robust feature detection and matching.

The observed performances suggest that while LIDAR-based methods generally provide more reliable and versatile mapping and trajectory estimations in complex and variable environments, there are niche scenarios where camera-based systems, particularly advanced ones like ORB-SLAM2, can outperform LIDAR-based SLAM systems. This underlines the importance of selecting the appropriate SLAM technology based on the specific requirements and characteristics of the environment in question, albeit with LIDAR-based SLAM systems performing better in outdoor environments.

5.5. Limitations

RTK-GNSS

By generating ground truth with RTK-GNSS with one single receiver, there are two challenges. Firstly, as we only have one antenna there is no way to get the orientation so instead of absolute pose error one can only consider absolute positional error for the collected dataset. Secondly, the accuracy of the RTK-GNSS receiver limits the precision level the estimated trajectories can provide, especially for the parking dataset as the covariance plots show larger errors in this sequence, and as such, some of the SLAM systems may actually provide a more accurate ground truth estimation than the RTK-GNSS itself.

Velocity limitation

The Robotnik platform has a maximum velocity of 5 km/h, and thus it is not possible to collect data at higher speeds than was done in this thesis. Higher speed would allow us to create datasets with larger variation and further explore to what degree velocities affect performance.

Processing delays

All LIDAR based systems were run with a playback rate of 1, meaning the rosbags were played back at the same rate as the original rosbags. However, ORB-SLAM2, ManhattanSLAM and RESLAM processed each image in the sequence, meaning that the actual runtime of these system could be longer than the duration of the rosbag. This allows these three systems to evaluate and generate an estimated pose for each image while the rest could process only as many images or scans as the computation would allow. A more comparative assessment would be to ensure that all systems process every image/scan or operate with a playback rate of 1.

This leads to a question about the performance of the computers, as better performance might reduce

the computation time for each system, allowing them to estimate more poses. More specifically, the virtual machine that the camera based SLAM systems were ran on, had no GPU passthrough such that the graphics card could be utilized for better SLAM performance. This was especially true for one of the SLAM systems, RGBDSLAMv2, which had the option to use SIFT-GPU. Another issue that was encountered with the VM, which was mentioned earlier, was the RAM problem that RESLAM faced when running it on the orchard dataset. In short, having more RAM allocated to the VM would make it possible for RESLAM to be fully evaluated on the orchard dataset. This could be solved by using either dual boot, a permanent install of an operating system, or simply acquiring and inserting more RAM into the host's PC.

Seasonal impact

Since the data was collected only during spring, there is limited seasonal variation in the data. This was mainly due to the fact that the Robotnik platform would struggle with grip during winter due to lack of winter tires.

5.6. Further works

Sensor fusion

An interesting future research could be to investigate how data fusion with both sensor technologies could improve the reconstructed map, specifically. This could be done by combining the resulting point cloud from both of the sensors into one. It is also possible to integrate an IMU sensor to improve the pose estimates which in turn could improve the reconstructed maps.

Multi-antenna RTK-GNSS

An immediate extension could involve the incorporation of a multi-antenna RTK-GNSS system to address the limitations identified with the single-receiver setup. A multi-antenna system would enable the measurement of orientation as well as position, thereby providing more data for evaluating SLAM systems.

Increased velocities

By conducting experiments at increased velocities, one could examine how velocity impacts the accuracy and reliability of SLAM systems. There is a lack of dataset with high velocity for RGB-D based systems, while there is a large gap in velocities between the collected dataset and KITTI. Moreover, understanding the velocity-dependent behavior of SLAM systems can lead to more robust designs that compensate for speed-induced errors.

Increased computational power

The computational aspects of SLAM systems, in particular the impact of hardware capabilities on per-

formance, could benefit from further exploration. Future studies could evaluate the benefits of running on a dedicated system, including GPU support. An empirical analysis of how processing power, memory bandwidth, and storage speeds affect the real-time capabilities of these systems could also provide insight into the optimal hardware configurations for various SLAM systems.

Collecting more data

Gathering data from both identical and different environments across various seasons, weather conditions, and times of day would yield an exceptionally varied dataset including an extensive spectrum of scenarios. Such a dataset would facilitate a comprehensive analysis of SLAM system performance, determining which configurations excel under specific conditions. This approach would significantly deepen our understanding of sensor-based SLAM systems' adaptability and robustness in diverse operational environments.

Parameter tuning

As this research did not change the inherent parameters for the environments, exploring with different parameters for each SLAM system and tuning them would allow for better performance for each of them, furthermore highlighting the strengths and weaknesses of each system. Experimenting with parameter tuning could showcase how each SLAM system performs with the optimal parameters for a given environment, providing a more extensive comparison across multiple scenarios. This could cause the results both within, and between each sensor-based SLAM system to be completely different than what was obtained in this research without parameter tuning.

Semantic SLAM

As all of the SLAM systems in this thesis, except for RESLAM, had the ability to extract a point cloud file. Post-processing on the reconstructed map could be explored, and as such, take a step into the direction of semantic SLAM. Semantic SLAM refers to the integration of semantic understanding with traditional SLAM algorithms, where the system not only maps an environment in terms of spatial layout but also interprets and classifies the elements within it [71]. This could involve identifying and labeling different objects and structures in the point clouds, such as distinguishing between trees, vehicles, buildings, and other landmarks. Furthermore, this could also have an impact on how dynamic objects are handled, thus making pose estimation more robust and accurate.

6. CONCLUSION

This thesis explored a comprehensive study on sensor-based SLAM systems, particularly examining the comparative effectiveness of LIDAR and camera-based SLAM systems within varied environmental conditions. The findings of this investigation underscore the complexity of pose estimation and mapping and the significant impact that sensor selection has on the performance of SLAM systems.

Throughout the study, LIDAR-based SLAM systems demonstrated superior performance in environments with complex geometries and variable lighting conditions, more specifically orchards and parking lots. This can be attributed to the LIDAR's ability to deliver accurate spatial data independent of ambient light conditions, thus ensuring robustness in mapping precision. Conversely, camera-based systems, while sometimes struggling under these conditions, showcased their potential in controlled, static environments where visual features are more prevalent and consistent. Notably, ORB-SLAM2 exhibited commendable performance in the parking lot dataset, highlighting the capabilities of advanced visual SLAM systems in certain areas.

The limitations identified through this research, such as the reliance on a single RTK-GNSS receiver for ground truth generation and the constraints imposed by the computational hardware, particularly absence of GPU passthrough and memory management, showcases how suboptimal experimental setups influence the performance of SLAM systems. These limitations not only highlight the need for a comprehensive approach to the evaluation of the SLAM system, but also underscore the potential inaccuracies that might arise when these issues are not addressed and dealt with.

Future work will benefit from a more integrative approach that harnesses the strengths of both LIDAR and camera technologies. The fusion of data from these sensors could lead to more comprehensive and accurate mapping capabilities. Additionally, the inclusion of semantic understanding in SLAM processes could aid in expanding the capabilities of these systems, enabling them not only to map, but also to understand and interact with their surroundings in a context-aware manner.

In conclusion, while this thesis has provided valuable insights into the performance of LIDAR and camera-based SLAM systems under various conditions, it also highlights the critical importance of continuing research in this field. As we move forward, the integration of both sensor fusion and semantic understanding in SLAM could be a valuable contribution and advancement to environmental mapping, creating systems that are robust in a multitude of environments and applications.

References

[1] Gao X, Zhang T. Introduction to Visual SLAM: From Theory to Practice. Springer Verlag; 2021.

[2] Mur-Artal R, Tardós JD. ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. IEEE Transactions on Robotics. 2017 Oct;33(5):1255-62.

[3] Endres F, Hess J, Sturm J, Cremers D, Burgard W. 3D Mapping with an RGB-D Camera. IEEE Transactions on Robotics. 2012 Jan;30(1):177 187.

[4] Labbé M, Michaud F. RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. Journal of Field Robotics. 2019;36(2):416-46.

[5] Schenk F, Fraundorfer F. RESLAM: A real-time robust edge-based SLAM system. In: 2019 International Conference on Robotics and Automation (ICRA); 2019. p. 154-60.

[6] Yunus R, Li Y, Tombari F. ManhattanSLAM: Robust Planar Tracking and Mapping Leveraging Mixture of Manhattan Frames. CoRR. 2021;abs/2103.15068. Available from: https://arxiv.org/abs/2103.15068.

[7] Intel Corporation. Intel RealSense Depth Camera D435i;. Accessed: 2024-04-14. https://www.intelrealsense.com/depth-camera-d435i/.

[8] sensor_msgs: sensor_msgs.point_cloud2 Namespace Reference;. Available from: https://docs.ros.org/en/melodic/api/sensor_msgs/html/namespacesensor__msgs_1_1point__cloud2.html.

[9] sensor-msgs/NavSatFix Documentation;. Available from: https://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/NavSatFix.html.

[10] Li ZX, Cui GH, Li CL, Zhang ZS. Comparative Study of Slam Algorithms for Mobile Robots in Complex Environment. In: 2021 6th International Conference on Control, Robotics and Cybernetics (CRC); 2021. p. 74-9.

[11] Zou Q, Sun Q, Chen L, Nie B, Li Q. A Comparative Analysis of LiDAR SLAM-Based Indoor Navigation for Autonomous Vehicles. IEEE Transactions on Intelligent Transportation Systems. 2022;23(7):6907-21.

[12] Filipenko M, Afanasyev I. Comparison of Various SLAM Systems for Mobile Robot in an Indoor Environment. In: 2018 International Conference on Intelligent Systems (IS); 2018. p. 400-7.

[13] Geiger A, Lenz P, Urtasun R. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In: Conference on Computer Vision and Pattern Recognition (CVPR); 2012. .

[14] Sturm J, Engelhard N, Endres F, Burgard W, Cremers D. A Benchmark for the Evaluation of RGB-D SLAM Systems. In: Proc. of the International Conference on Intelligent Robot Systems (IROS); 2012. .

[15] Thrun S, Burgard W, Fox D. Probabilistic Robotics (Intelligent Robotics and Autonomous Agents). The MIT Press; 2005.

[16] Lowe DG. Distinctive Image Features from Scale-Invariant Keypoints. vol. 60. Springer; 2004. p. 91-110.

[17] Bay H, Tuytelaars T, Van Gool L. SURF: Speeded Up Robust Features. In: Leonardis A, Bischof H, Pinz A, editors. Computer Vision – ECCV 2006. vol. 3951 of Lecture Notes in Computer Science. Springer, Berlin, Heidelberg; 2006. p. 404-17.

[18] Rublee E, Rabaud V, Konolige K, Bradski G. ORB: An efficient alternative to SIFT or SURF. In: 2011 International Conference on Computer Vision; 2011. p. 2564-71.

[19] Fischler MA, Bolles RC. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun ACM. 1981 jun;24(6):381–395. Available from: https://doi.org/10.1145/358669.358692.

[20] Besl PJ, McKay ND. A method for registration of 3-D shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence. 1992;14(2):239-56.

[21] Point Cloud Library. PassThrough Filtering;. Accessed: 2024-04-14. https://pointclouds.org/documentation/tutorials/passthrough.html.

[22] Point Cloud Library. Voxel Grid Downsampling;. Accessed: 2024-04-14. https://pcl.readthedocs.io/projects/tutorials/en/latest/voxel_grid.html.

[23] Downsample a 3-D point cloud - MATLAB pcdownsample - MathWorks Nordic;. Available from: https://se.mathworks.com/help/vision/ref/pcdownsample.html.

[24] Hornung A, Wurm KM, Bennewitz M, Stachniss C, Burgard W. OctoMap: an efficient probabilistic 3D mapping framework based on octrees. Autonomous Robots. 2013 Apr;34(3):189-206. Available from: http://link.springer.com/10.1007/s10514-012-9321-0.

[25] Grisetti G, Kummerle R, Stachniss C, Burgard W. A Tutorial on Graph-Based SLAM. IEEE Intelligent Transportation Systems Magazine. 2010;2(4):31-43. Available from: http://ieeexplore.ieee.org/document/5681215/.

[26] Azzam R, Taha T, Huang S, et al. Feature-based visual simultaneous localization and mapping: a survey. vol. 2. Springer; 2020. p. 224.

[27] Sahili AR, Hassan S, Sakhrieh SM, Mounsef J, Maalouf N, Arain B, et al. A Survey of Visual SLAM Methods. IEEE Access. 2023;11:139643-77.

[28] Mur-Artal R, Montiel JMM, Tardos JD. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. IEEE Transactions on Robotics. 2015 Oct;31(5):1147–1163. Available from: http://dx.doi.org/10.1109/TRO.2015.2463671.

[29] Luo K, Lin M, Wang P, Zhou S, Yin D, Zhang HL. Improved ORB-SLAM2 Algorithm Based on Information Entropy and Image Sharpening Adjustment. Mathematical Problems in Engineering. 2020 09;2020:13.

[30] Barros, Andréa Macario and Michel, Maugan and Moline, Yoann and Corre, Gwenolé and Carrel, Frédérick. A Comprehensive Survey of Visual SLAM Algorithms. Robotics. 2023;11(1):24.

[31] Coughlan J, Yuille A. The Manhattan World Assumption: Regularities in scene statistics which enable Bayesian inference. NIPS. 1970 02.

[32] szenergy/awesome-lidar. szenergy; 2024. Original-date: 2020-04-13T14:15:48Z. Available from: https://github.com/szenergy/awesome-lidar.

[33] Zhang J, Singh S. LOAM: Lidar Odometry and Mapping in Real-time. In: Robotics: Science and Systems X. Robotics: Science and Systems Foundation; 2014. Available from: http://www.roboticsproceedings.org/rss10/p07.pdf.

[34] Eigen Developers. Eigen: a C++ template library for linear algebra;. Accessed: 2024-04-14. https://eigen.tuxfamily.org/index.php?title=Main_Page.

[35] Ceres Solver Developers. Ceres Solver;. Accessed: 2024-04-14. http://ceres-solver.org/.

[36] HKUST-Aerial-Robotics/A-LOAM: Advanced implementation of LOAM;. Available from: https://github.com/HKUST-Aerial-Robotics/A-LOAM.

[37] Wang H, Wang C, Chen CL, Xie L. F-LOAM: Fast LiDAR Odometry And Mapping. In: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS); 2021. p. 4390-6. ArXiv:2107.00822 [cs]. Available from: http://arxiv.org/abs/2107.00822.

[38] Kim G, Choi S, Kim A. Scan Context++: Structural Place Recognition Robust to Rotation and Lateral Variations in Urban Environments. arXiv; 2021. ArXiv:2109.13494 [cs]. Available from: http://arxiv.org/abs/2109.13494.

[39] Shan T, Englot B. LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Madrid: IEEE; 2018. p. 4758-65. Available from: https://ieeexplore.ieee.org/document/8594299/.

[40] koide3. koide3/hdl_graph_slam; 2024. Original-date: 2018-01-01T07:35:43Z. Available from: https://github.com/koide3/hdl_graph_slam.

[41] Koide K, Miura J, Menegatti E. A portable three-dimensional LIDAR-based system for long-term and wide-area people behavior measurement. International Journal of Advanced Robotic Systems. 2019 Mar;16(2):172988141984153. Available from: http://journals.sagepub.com/doi/10.1177/1729881419841532.

[42] Akai N, Morales LY, Takeuchi E, Yoshihara Y, Ninomiya Y. Robust localization using 3D NDT scan matching with experimentally determined uncertainty and road marker matching. In: 2017 IEEE Intelligent Vehicles Symposium (IV); 2017. p. 1356-63.

[43] Vizzo I, Guadagnino T, Mersch B, Wiesmann L, Behley J, Stachniss C. KISS-ICP: In Defense of Point-to-Point ICP – Simple, Accurate, and Robust Registration If Done the Right Way. IEEE Robotics and Automation Letters. 2023 Feb;8(2):1029-36. Available from: https://ieeexplore.ieee.org/document/10015694,https://github.com/PRBonn/kiss-icp/tree/v0.3.0.

[44] FRAMOS. What Are Depth Sensing Cameras and How Do They Work?;. Accessed: 2024-02-07. http://framos.com/en/articles/what-are-depth-sensing-cameras-and-how-do-they-work.

[45] Li Y, Ibanez-Guzman J. Lidar for Autonomous Driving: The principles, challenges, and trends for automotive lidar and perception systems. IEEE Signal Processing Magazine. 2020 Jul;37(4):50-61. ArXiv:2004.08467 [cs]. Available from: http://arxiv.org/abs/2004.08467.

[46] OS0-128: 128-CHANNEL ULTRA-WIDE DIGITAL LIDAR SENSOR;. Available from: https://atyges.es/en/tienda/product/os0-128/.

[47] Quigley M, Gerkey B, Conley K, Faust J, Foote T, Leibs J, et al. ROS: an open-source Robot Operating System.

[48] sensor-msgs/PointCloud2 Documentation;. Available from: https://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/PointCloud2.html.

[49] ROS Wiki Contributors. rosbag;. Accessed: 07.02.2024. http://wiki.ros.org/rosbag.

[50] ROS Wiki Contributors. rviz;. Accessed: 07.02.2024. http://wiki.ros.org/rviz.

[51] Kim E, Kim Sk. Global Navigation Satellite System Real-Time Kinematic Positioning Framework for Precise Operation of a Swarm of Moving Vehicles. Sensors. 2022;22(20). Available from: https://www.mdpi.com/1424-8220/22/20/7939.

[52] Grupp M. evo: Python package for the evaluation of odometry and SLAM.; 2017. https://github.com/MichaelGrupp/evo.

[53] Umeyama S. Least-squares estimation of transformation parameters between two point patterns. IEEE Transactions on Pattern Analysis and Machine Intelligence. 1991;13(4):376-80.

[54] Sturm J, Burgard W, Cremers D. Evaluating Egomotion and Structure-from-Motion Approaches Using the TUM RGB-D Benchmark. 2012 01.

[55] RB-VOGUI Mobile Robot - Outdoor Mobile Robot | Robotnik®;. Available from: https://robotnik.eu/products/mobile-robots/rb-vogui-en/.

[56] Oracle. Oracle VM VirtualBox; 2024. Virtualization software. Available from: https://www.virtualbox.org/.

[57] Docker: Accelerated Container Application Development; 2022. Available from: https://www.docker.com/.

[58] CloudCompare - Open Source project;. Available from: https://www.danielgm.net/cc/.

[59] Google Earth. Top down image of NMBU campus; 2024. Available from: https://www.google.com/earth/,https://earth.google.com/earth/d/17xgDlVixp73cj50sPS9ISHeTXEexwPBo?usp=sharing.

[60] Charles F F Karney. geographiclib;. Accessed: 2024-05-10. https://geographiclib.sourceforge.io/html/python/.

[61] Xiang G. ORBSLAM2 with pointcloud map. GitHub;. https://github.com/gaoxiang12/ORBSLAM2_with_pointcloud_map.

[62] IntRoLab. RTAB-Map's ROS package. GitHub;. https://github.com/introlab/rtabmap_ros.

[63] Endres F. RGB-D SLAM for ROS. GitHub;. https://github.com/felixendres/rgbdslam_v2.

[64] Yunus R. ManhattanSLAM. GitHub;. https://github.com/razayunus/ManhattanSLAM.

[65] Schenk F. RESLAM: A real-time robust edge-based SLAM system. GitHub;. https://github.com/fabianschenk/RESLAM.

[66] Computer Vision Group T. Useful Tools for the RGB-D Benchmark; 2024. Accessed: 2024-04-21. https://cvg.cit.tum.de/data/datasets/rgbd-dataset/tools.

[67] wh200720041/floam: Fast LOAM: Fast and Optimized Lidar Odometry And Mapping for indoor/outdoor localization IROS 2021;. Available from: https://github.com/wh200720041/floam.

[68] Kim G. gisbi-kim/SC-A-LOAM; 2024. Original-date: 2021-05-10T12:29:05Z. Available from: https://github.com/gisbi-kim/SC-A-LOAM.

[69] Kim D. kimdaebeom/LeGO-LOAM; 2022. Original-date: 2022-01-28T06:18:17Z. Available from: https://github.com/kimdaebeom/LeGO-LOAM.

[70] Zhou QY, Park J, Koltun V. Open3D: A modern library for 3D data processing. arXiv:180109847. 2018.

[71] Yu C, Liu Z, Liu XJ, Xie F, Yang Y, Wei Q, et al. DS-SLAM: A Semantic Visual SLAM towards Dynamic Environments. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Madrid: IEEE; 2018. p. 1168-74. Available from: https://ieeexplore.ieee.org/document/8593691/.

[72] Cheng R. bag file to image files in tum dataset and realsense dataset; 2024. Accessed: 2024-04-12. https://gist.github.com/rancheng/11b13ec6c2182781af7fdc11ed7c3c18.

7. APPENDIX

Camera configuration files

ORB-SLAM2 and Manhattan-SLAM

```
%YAML:1.0

#--------------------------------------------------------------------------------------------
# Camera Parameters. Adjust them!
#--------------------------------------------------------------------------------------------

# Camera calibration and distortion parameters (OpenCV)
Camera.fx: 605.612060546875
Camera.fy: 604.5374145507812
Camera.cx: 322.8209228515625
Camera.cy: 248.99522399902344

Camera.k1: 0.0
Camera.k2: 0.0
Camera.p1: 0.0
Camera.p2: 0.0

Camera.width: 640
Camera.height: 480

# Camera frames per second
Camera.fps: 30.0

# IR projector baseline times fx (aprox.)
Camera.bf: 30.2806

# Color order of the images (0: BGR, 1: RGB. It is ignored if images
 ↪  are grayscale)
Camera.RGB: 1

# Close/Far threshold. Baseline times.
```

```
ThDepth: 50.0

# Deptmap values factor
DepthMapFactor: 1000.0

#--------------------------------------------------------------------------
# ORB Parameters
#--------------------------------------------------------------------------

# ORB Extractor: Number of features per image
ORBextractor.nFeatures: 1000

# ORB Extractor: Scale factor between levels in the scale pyramid
 ↪
ORBextractor.scaleFactor: 1.2

# ORB Extractor: Number of levels in the scale pyramid
ORBextractor.nLevels: 8

# ORB Extractor: Fast threshold
# Image is divided in a grid. At each cell FAST are extracted imposing
 ↪  a minimum response.
# Firstly we impose iniThFAST. If no corners are detected we impose a
 ↪  lower value minThFAST
# You can lower these values if your images have low
 ↪  contrast
ORBextractor.iniThFAST: 20
ORBextractor.minThFAST: 7

#--------------------------------------------------------------------------
# Viewer Parameters
#--------------------------------------------------------------------------
Viewer.KeyFrameSize: 0.05
Viewer.KeyFrameLineWidth: 1
Viewer.GraphLineWidth: 0.9
Viewer.PointSize: 2
```

```
Viewer.CameraSize: 0.08
Viewer.CameraLineWidth: 3
Viewer.ViewpointX: 0
Viewer.ViewpointY: -0.7
Viewer.ViewpointZ: -1.8
Viewer.ViewpointF: 500

#--------------------------------------------------------------------------
# PointCloud Mapping
#--------------------------------------------------------------------------
PointCloudMapping.Resolution: 0.04
```

RESLAM

```
%YAML:1.0

#--------------------------------------------------------------------------
# Camera Parameters. Adjust them!
#--------------------------------------------------------------------------


#Sensor settings
InputSensorType: 0
InputDepthScaleFactor: 5000.0
IntrinsicsImgSize.width: 640
IntrinsicsImgSize.height: 480
# Camera calibration and distortion parameters (OpenCV)
IntrinsicsCamFx: 605.612060546875
IntrinsicsCamFy: 604.5374145507812
IntrinsicsCamCx: 322.8209228515625
IntrinsicsCamCy: 248.99522399902344
IntrinsicsCamK1: 0
IntrinsicsCamK2: 0
IntrinsicsCamP1: 0
IntrinsicsCamP2: 0
IntrinsicsCamK3: 0
```

```
InputCannyEdgeTh1: 60
InputCannyEdgeTh2: 40
InputSmoothEdgeImage: 0
InputComputeGradientsForEdgeDetector: 0
InputDatasetFile: "associations.txt"
InputDatasetFolder: "/home/marko/Music/parking_tum_sync_tum"
InputReadGT: 1 #groundtruth.txt
InputReadNFrames: 5000
InputDepthMin: 0.1
InputDepthMax: 5.2
InputColorFormatRGB: 0
#Input
InputSkipFirstNFrames: 0
```

RESLAM

```
%YAML:1.0
#GENERAL
SystemMultiThreading: 0
ViewerPointSize: 2

#Tracker
# TrackerNPyrLevels: 3
TrackerEvalLvlForInit: 2
TrackerSkipResdiualsOnLowerLvls: 0
TrackerTrackFromFrameToKf: 0 #leave it deactivated at the moment!
TrackerUseWeightsResidualsForErrorComputation: 1
TrackerResidualHuberWeight: 0.3
TrackerUseEdgeWeight: 0
TrackerUseEdgeFilter: 1
TrackerOpticalFlowTFactor: 0.1
TrackerOpticalFlowRTFactor: 0.15
TrackerOpticalFlowThreshold: 1
TrackerHistDist: 1.5 #For LC
TrackerAvgResidualBeforeTrackingLoss: 2.5
```

```
#Output Settings
OutputPoses: 1
OutputRecordImages: 0
OutputPoseFileFolder: "/home/marko/RESLAM/output"

#Local Mapper
EnableLocalMapping: 1
LocalMapperUseCoarseDistanceMap: 1
LocalMapperOptimizeDepth: 1
LocalMapperOptimizeInitDepth: 1
LocalMapperConditionalDepthOptimization: 1
LocalMapperMaxDistForValidPixels: 5.0 #5.0
LocalMapperMinDistInDistMapForValidPixels: 5.0 #7.0
LocalMapperShowResidualsInImage: 0
LocalMapperEdgeDistance: 5.0
LocalMapperLinearProcessing: 1
LocalMapperDoMarginalize: 1
LocalMapperAdaptMinActDist: 0
LocalMapperUseNewPoseUpdate: 0

#Loop Closure
EnableLoopClosure: 0
# This sets the minimum distance between two candidates, where we
 ↪  don't close a loop
LoopClosureMinDistBetweenFrames: 15
LoopClosureFramesToCheck: 3
LoopClosureEdgeWeight: 3 #Insert loop 3 times
LoopClosureUseBAConstraints: 1
LoopClosureLinearProcessing: 1
LoopClosureKeyframeThreshold: 0.25
LoopClosureNumberOfBAConstraints: 3
LoopClosureDoNotCloseLoopsIfClosedLastN: 7
LoopClosureOnlyUseFernKF: 1
LoopClosureFixWindowPoses: 1
LoopClosureAddAdjacentConstraints: 0
LoopClosureDoBackCheck: 1
```

```
#Relocaliser
EnableRelocaliser: 1

FernDatabaseAlwaysAddKf: 0
#FernDatabaseHarvestingThreshold: 0.2
#FernDatabaseNumFerns: 500
#FernDatabaseNumDecisions: 4

#Debug
SystemDebugTextOutput: 0
TrackerShowIterationsDebug: 0
TrackerShowInitDebug: 0
LoopClosureDebugShowTransformationImages: 0
LoopClosureTrackerShowIterations: 0
```

RTAB-Map

```
<?xml version="1.0"?>
  <launch>
  <param name="use_sim_time" type="bool" value="True"/>

  <node pkg="tf" type="static_transform_publisher"
   ↪  name="camera_base_link"
        args="0 0 0 0 0 0 1 base_link
         ↪  robot_front_rgbd_camera_color_optical_frame 100" />

  <group ns="rtabmap">

    <!-- Odometry -->
    <node pkg="rtabmap_odom" type="rgbd_odometry" name="rgbd_odometry"
     ↪  output="screen">
      <remap from="rgb/image"
       ↪  to="/robot/camera/color/image_raw"/>
      <remap from="depth/image"
       ↪  to="/robot/camera/depth/image_rect_raw"/>
      <remap from="rgb/camera_info"
       ↪  to="/robot/camera/color/camera_info"/>
```

```
    <param name="frame_id" type="string" value="base_link"/>
  </node>

  <!-- SLAM -->
  <node name="rtabmap" pkg="rtabmap_slam" type="rtabmap"
   ↪  output="screen" args="--delete_db_on_start">
    <param name="subscribe_depth" type="bool" value="true"/>
    <param name="frame_id" type="string" value="base_link"/>

    <param name="Rtabmap/StartNewMapOnLoopClosure" type="string"
     ↪  value="true"/>
    <param name="RGBD/CreateOccupancyGrid" type="string"
     ↪  value="false"/>
    <param name="Rtabmap/CreateIntermediateNodes" type="string"
     ↪  value="true"/>
    <param name="RGBD/LinearUpdate" type="string" value="0"/>
    <param name="RGBD/AngularUpdate" type="string" value="0"/>

    <remap from="rgb/image"
     ↪  to="/robot/camera/color/image_raw"/>
    <remap from="depth/image"
     ↪  to="/robot/camera/depth/image_rect_raw"/>
    <remap from="rgb/camera_info"
     ↪  to="/robot/camera/color/camera_info"/>
    <remap from="odom" to="odom"/>
  </node>

</group>
</launch>
```

RGBDSLAMv2

```
<launch>
  <!-- These parameters are used for the benchmark evaluation.
   ↪  Documentation can be found in parameter_server.cpp -->
  <!--<env name="ROSCONSOLE_CONFIG_FILE" value="$(find
   ↪  rgbdslam)/rgbd_benchmark/log_eval.conf"/>-->
```

```
<env name="ROSCONSOLE_FORMAT" value="[${severity}] Time:[${time}]
 ↪  Thread:[${thread}]: ${message}"/>
<arg name="debug" default="false"/>
<!--arg if="$(arg debug)" name="launch_prefix"
 ↪  value="/usr/bin/time"/-->
<!--arg if="$(arg debug)" name="launch_prefix" value="/usr/bin/gdb
 ↪  -ex run -args"/-->
<arg if="$(arg debug)" name="launch_prefix" value="/usr/bin/xterm -e
 ↪  gdb -ex run -args"/>
<!--arg if="$(arg debug)" name="launch_prefix" value="/usr/bin/ddd
 ↪  -ex run -args"/-->
<!--arg if="$(arg debug)" name="launch_prefix"
 ↪  value="/usr/bin/valgrind -DELETEME-tool=callgrind
 ↪  -DELETEME-callgrind-out-file=/tmp/callgrind.out"/-->
<!--arg if="$(arg debug)" name="launch_prefix"
 ↪  value="/usr/bin/valgrind -DELETEME-leak-check=full "/-->
<arg unless="$(arg debug)" name="launch_prefix" value=""/>

<!-- These parameters are defined in the calling script
 ↪  (run_tests.sh)-->
<arg name="bagfile_name" />
<arg name="feature_type" default="ORB"/>
<arg name="ransac_iterations" default="1500"/>
<arg name="optimizer_skip_step" default="1"/>
<arg name="observability_threshold" default="0"/>
<arg name="use_root_sift" default="false"/>
<arg name="max_keypoints" default="500"/>
<arg name="match_candidates" default="4"/>
<arg name="sampled_candidates" default="4"/>
<arg name="gui" default="true"/>


<node pkg="rgbdslam" type="rgbdslam" name="rgbdslam" required="true"
 ↪  output="screen"  launch-prefix="$(arg launch_prefix)">
  <!-- Input data settings-->
```

```
<param name="config/topic_image_mono"
 ↪  value="/robot/camera/color/image_raw"/>
<param name="config/topic_image_depth"
 ↪  value="/robot/camera/depth/image_rect_raw"/>
<param name="config/camera_info_topic"
 ↪  value="/robot/camera/color/camera_info"/>


<param name="config/depth_camera_fx"
 ↪  value="605.612060546875"/>
<param name="config/depth_camera_fy"
 ↪  value="604.5374145507812"/>
<param name="config/depth_camera_cx"
 ↪  value="322.8209228515625"/>
<param name="config/depth_camera_cy"
 ↪  value="248.99522399902344"/>


<!-- TF information settings -->
<param name="config/fixed_frame_name"              value="/map"/>
<param name="config/odom_frame_name"               value=""/>
<param name="config/ground_truth_frame_name"
 ↪  value=""/><!--empty string if no ground truth-->
<param name="config/base_frame_name"
 ↪  value="robot_front_rgbd_camera_color_optical_frame"/> <!--
 ↪  /openni_camera for hand-held kinect. For robot, e.g.,
 ↪  /base_link -->
<param name="config/fixed_camera"                  value="true"/>
 ↪  <!--is the kinect fixed with respect to base, or can it be
 ↪  moved (false makes sense only if transform betw. base_frame
 ↪  and openni_camera is sent via tf)-->
<param name="config/start_paused"                  value="false"/>
<param name="config/store_pointclouds"             value="true"/>
 ↪  <!-- if, e.g., only trajectory is required, setting this to
 ↪  false saves lots of memory -->
```

```
<param name="config/feature_detector_type"          value="$(arg
 ↪  feature_type)"/>
<param name="config/feature_extractor_type"         value="$(arg
 ↪  feature_type)"/>

<param name="config/matcher_type"
 ↪  value="BRUTEFORCE"/> <!-- FLANN (not avail for ORB features),
 ↪  SIFTGPU (only for SIFTGPU detector)  or BRUTEFORCE-->
      <param name="config/bagfile_name"
       ↪  value="$(arg bagfile_name)"/>
<!--param name="config/nn_distance_ratio"
 ↪  value="0.80"/--> <!-- Feature correspondence is valid if
 ↪  distance to nearest neighbour is smaller than this parameter
 ↪  times the distance to the 2nd neighbour -->
      <param name="config/max_keypoints"
       ↪  value="$(arg max_keypoints)"/>
      <param name="config/min_keypoints"
       ↪  value="0"/>
<param name="config/sufficient_matches"
 ↪  value="400"/><!-- Instead of matching all new descriptors
 ↪  against all of a previous node in one step,
 ↪  sufficient_matches+100 of the new descriptors are iteratively
 ↪  compared to all of the previous node until sufficient_matches
 ↪  are found. Setting this parameter low (e.g. 2x min_matches)
 ↪  speeds up comparisons of frames with many matches, but with a
 ↪  potential loss of accuracy, as the transformation is estimated
 ↪  from less features. Set it to max_keypoints for maximum
 ↪  accuracy -->
<param name="config/min_translation_meter"
 ↪  value="-1.0"/><!--disabled -->
<param name="config/min_rotation_degree"
 ↪  value="-1.0"/><!--disabled -->
<param name="config/max_translation_meter"
 ↪  value="100.0"/><!-- transformations with motion more than this
 ↪  per second!, will be omitted -->
```

```
<param name="config/max_rotation_degree"
 ↪  value="900"/><!-- transformations with motion more than this
 ↪  per second!, will be omitted -->
      <param name="config/min_time_reported"
       ↪  value="0.02"/>

<param name="config/predecessor_candidates"        value="$(arg
 ↪  match_candidates)"/><!--search through this many immediate
 ↪  predecessor nodes for corrspondences -->
<param name="config/neighbor_candidates"           value="$(arg
 ↪  match_candidates)"/><!--search through this many graph
 ↪  neighbour nodes for corrspondences -->
<param name="config/min_sampled_candidates"        value="$(arg
 ↪  sampled_candidates)"/><!--search through this many uniformly
 ↪  sampled nodes for corrspondences -->
<param name="config/max_connections"
 ↪  value="-1"/><!-- stop after this many succesfully found
 ↪  spation relations -->

      <param name="config/drop_async_frames"
       ↪  value="false"/>
      <param name="config/min_matches"
       ↪  value="20"/>
      <param name="config/max_dist_for_inliers"
       ↪  value="0.5"/>
      <param name="config/ransac_iterations"
       ↪  value="$(arg ransac_iterations)"/><!-- these are fast,
       ↪  so high values are ok -->
      <param name="config/use_gui"
       ↪  value="$(arg gui)"/>
      <param name="config/use_glwidget"
       ↪  value="$(arg gui)"/>
      <param name="config/concurrent_node_construction"
       ↪  value="true"/>
      <param name="config/concurrent_edge_construction"
       ↪  value="true"/>
```

```
      <param name="config/concurrent_optimization"
       ↪  value="true"/>
<param name="config/optimizer_skip_step"           value="$(arg
 ↪  optimizer_skip_step)"/><!--optimize at end only -->
<param name="config/backend_solver"                value="pcg"/>
<param name="config/pose_relative_to"
 ↪  value="inaffected"/>
<param name="config/optimize_landmarks"            value="true"/>
 ↪  <!-- Include feature poses as vertices in graph
 ↪  optimization-->
      <param name="config/data_skip_step"
       ↪  value="2"/><!-- skip every n-th frame completely -->
      <param name="config/visualization_skip_step"
       ↪  value="1"/> <!-- draw only every nth pointcloud row and
       ↪  line, high values require higher
       ↪  squared_meshing_threshold -->
      <param name="config/encoding_bgr"
       ↪  value="false"/>
      <param name="config/visualize_mono_depth_overlay"
       ↪  value="false"/>
<param name="config/squared_meshing_threshold"
 ↪  value="0.0081"/>
<param name="config/batch_processing"              value="true"/>
 ↪  <!--store results and close after bagfile has been
 ↪  processed-->
<param name="config/keep_all_nodes"                value="true"/>
 ↪  <!-- add nodes with constant motion if no transformation can
 ↪  be found -->
<param name="config/use_icp"                       value="false"/>
 ↪  <!-- Ignored if ICP is not compiled in (see top of
 ↪  CMakeLists.txt) -->
<param name="config/gicp_max_cloud_size"           value="15000"/>
<param name="config/use_root_sift"                 value="$(arg
 ↪  use_root_sift)"/>
```

```
    <param name="config/optimizer_iterations"
     ↪  value="0.001"/><!-- maximum of iterations in online operation
     ↪  (i.e., does affect the final optimization in batch mode) -->
    <param name="config/emm__skip_step"                value="1"/>
    <param name="config/emm__mark_outliers"            value="false"/>
    <param name="config/observability_threshold"       value="$(arg
     ↪  observability_threshold)"/>
  <!--      <param name="config/max_edge_error"       value="0.01"/>
   ↪  -->
    <param name="config/cloud_creation_skip_step"      value="8"/>
     ↪  <!-- Only active if cloud is computed (i.e. "topic_points" is
     ↪  empty. This value multiplies to emm__skip_step and
     ↪  visualization_skip_step-->
    <param name="config/octomap_resolution"            value="0.01"/>
     ↪  <!-- Only active if cloud is computed (i.e. "topic_points" is
     ↪  empty. This value multiplies to emm__skip_step and
     ↪  visualization_skip_step-->
    <!--param name="config/g2o_transformation_refinement"
     ↪  value="1000"/--> <!-- Use g2o to refine the ransac result,
     ↪  i.e. optimize the Mahalanobis distance in a final step.-->
  </node>
</launch>
```

Launch files for KITTI

A-LOAM

```
<launch>
    <arg name="sequence" default="sequence05"/>
    <arg name="speed" default="speed01"/>
    <arg name="record" default="false"/>


    <node pkg="rosbag" type="play" name="rosbag_play" args="--clock
     ↪  -r 1 -k
     ↪  /home/tor/Downloads/semantikitti2bag/semantickitti_$(arg
     ↪  sequence).bag" required="true" />
```

```
<group if="$(arg record)">
    <node name="rosbag_record" pkg="rosbag" type="record"
    args="/odom_pose /aft_mapped_to_init -O '/mnt/d/OneDrive -
     ↪  Norwegian University of Life Sciences/master/results/$(arg
     ↪  sequence)/aloam_$(arg sequence)_$(arg speed).bag'"
    output="screen"/>
</group>

<param name="/use_sim_time" value="true" />

<param name="scan_line" type="int" value="64" />

<!-- if 1, do mapping 10 Hz, if 2, do mapping 5 Hz. Suggest to use
 ↪  1, it will adjust frequence automaticlly -->
<param name="mapping_skip_frame" type="int" value="1" />

<!-- remove too closed points -->
<param name="minimum_range" type="double" value="5"/>


<param name="mapping_line_resolution" type="double" value="0.4"/>
<param name="mapping_plane_resolution" type="double" value="0.8"/>

<node pkg="aloam_velodyne" type="ascanRegistration"
 ↪  name="ascanRegistration" output="screen" />

<node pkg="aloam_velodyne" type="alaserOdometry"
 ↪  name="alaserOdometry" output="screen" />

<node pkg="aloam_velodyne" type="alaserMapping"
 ↪  name="alaserMapping" output="screen" />

<arg name="rviz" default="true" />
<group if="$(arg rviz)">
```

```
        <node launch-prefix="nice" pkg="rviz" type="rviz" name="rviz"
        ↪  args="-d $(find
        ↪  aloam_velodyne)/rviz_cfg/aloam_velodyne.rviz" />
    </group>

</launch>
```

F-LOAM

```
<?xml version="1.0"?>
<launch>
    <arg name="sequence" default="sequence05"/>
    <arg name="speed" default="speed01"/>
    <arg name="record" default="false"/>


    <node pkg="rosbag" type="play" name="rosbag_play" args="--clock
    ↪  -r 1 -k
    ↪  /home/tor/Downloads/semantikitti2bag/semantickitti_$(arg
    ↪  sequence).bag" required="true" />

    <group if="$(arg record)">
    <node name = "rosbag_record" pkg="rosbag" type="record"
    args="/odom_pose /odom -O '/mnt/d/OneDrive - Norwegian University
    ↪  of Life Sciences/master/results/$(arg sequence)/floam_$(arg
    ↪  sequence)_$(arg speed).bag'"
    output="screen"/>
    </group>

    <!-- For Velodyne HDL-64 -->
    <param name="scan_line" value="64" />

    <!--- Sim Time -->
    <param name="/use_sim_time" value="true" />
    <param name="scan_period" value="0.1" />

    <param name="vertical_angle" type="double" value="2.0" />
```

```
<param name="max_dis" type="double" value="90.0" />
<param name="map_resolution" type="double" value="0.4" />
<param name="min_dis" type="double" value="3.0" />
<!--- -->
<node pkg="floam" type="floam_odom_estimation_node"
 ↪  name="floam_odom_estimation_node" output="screen"/>
<node pkg="floam" type="floam_laser_processing_node"
 ↪  name="floam_laser_processing_node" output="screen"/>
<node pkg="floam" type="floam_laser_mapping_node"
 ↪  name="floam_laser_mapping_node" output="screen"/>

<node pkg="tf" type="static_transform_publisher"
 ↪  name="word2map_tf"  args="0 0 0 0 0 0 /world /map 10" />

<arg name="rviz" default="true" />
<group if="$(arg rviz)">
    <node launch-prefix="nice" pkg="rviz" type="rviz" name="rviz"
     ↪  args="-d $(find floam)/rviz/floam_mapping.rviz" />
</group>


      <node pkg="hector_trajectory_server"
       ↪  type="hector_trajectory_server"
       ↪  name="trajectory_server_loam" ns="gt" >
    <param name="/target_frame_name" value="world" />
    <param name="/source_frame_name" value="velodyne" />
    <param name="/trajectory_update_rate" value="10.0" />
    <param name="/trajectory_publish_rate" value="10.0" />
</node>
<node pkg="hector_trajectory_server"
 ↪  type="hector_trajectory_server" name="trajectory_server_loam"
 ↪  ns="base_link" >
    <param name="/target_frame_name" value="world" />
    <param name="/source_frame_name" value="base_link" />
    <param name="/trajectory_update_rate" value="10.0" />
    <param name="/trajectory_publish_rate" value="10.0" />
```

```
    </node>

</launch>


    <!--TO SAVE THE RESULTING MAP RUN
        rosrun pcl_ros pointcloud_to_pcd input:=/map
    -->
```

HDL-Graph SLAM

```
<?xml version="1.0"?>
<launch>
    <arg name="sequence" default="sequence05"/>
    <arg name="speed" default="speed01"/>
    <arg name="record" default="false"/>


    <node pkg="rosbag" type="play" name="rosbag_play" args="--clock
     ↪  -r 1 -k
     ↪  /home/tor/Downloads/semantikitti2bag/semantickitti_$(arg
     ↪  sequence).bag" required="true" />

    <group if="$(arg record)">
    <node name = "rosbag_record" pkg="rosbag" type="record"
    args="/odom_pose /odom -O '/mnt/d/OneDrive - Norwegian University
     ↪  of Life Sciences/master/results/$(arg sequence)/hdl_$(arg
     ↪  sequence)_$(arg speed).bag'"
    output="screen"/>
    </group>


    <arg name="rviz" default="true" />
    <arg name="rviz_config" default="$(find
     ↪  hdl_graph_slam)/rviz/hdl_graph_slam.rviz" />
```

```
    <rosparam param="use_sim_time">true</rosparam>



    <include file="$(find
     ↪  hdl_graph_slam)/launch/hdl_graph_slam_kitti.launch"/>

    <group if="$(arg rviz)">
        <node pkg="rviz" type="rviz" name="rviz" args="-d $(arg
         ↪  rviz_config)" />
    </group>

</launch>
```

KISS-ICP

```
<?xml version="1.0"?>
<launch>
    <arg name="sequence" default="sequence05"/>
    <arg name="speed" default="speed01"/>
    <arg name="record" default="false"/>

    <node pkg="rosbag" type="play" name="rosbag_play" args="--clock
     ↪  -r 1 -k
     ↪  /home/tor/Downloads/semantikitti2bag/semantickitti_$(arg
     ↪  sequence).bag" required="true" />

    <group if="$(arg record)">
    <node name="rosbag_record" pkg="rosbag" type="record"
    args="/odom_pose /kiss/odometry  -O '/mnt/d/OneDrive - Norwegian
     ↪  University of Life Sciences/master/results/$(arg
     ↪  sequence)/kiss_icp_$(arg sequence)_$(arg speed).bag'"
    output="screen"/>
    </group>

    <include file="$(find kiss_icp)/launch/odometry.launch">
        <arg name="topic" value="velodyne_points"/>
```

```
        <arg name="odom_frame" value="map"/>
    </include>


</launch>
```

LeGO-LOAM

```
<?xml version="1.0"?>
<launch>
    <arg name="sequence" default="sequence05"/>
    <arg name="speed" default="speed01"/>
    <arg name="record" default="false"/>

    <node pkg="rosbag" type="play" name="rosbag_play" args="--clock
     ↪  -r 1 -k
     ↪  /home/tor/Downloads/semantikitti2bag/semantickitti_$(arg
     ↪  sequence).bag" required="true" />


    <group if="$(arg record)">
    <node name="rosbag_record" pkg="rosbag" type="record"
    args="/odom_pose /aft_mapped_to_init  -O '/mnt/d/OneDrive -
     ↪  Norwegian University of Life Sciences/master/results/$(arg
     ↪  sequence)/lego_$(arg sequence)_$(arg speed).bag'"
    output="screen"/>
    </group>


    <include file="$(find lego_loam)/launch/run.launch">
    </include>



</launch>
```

SC-A-LOAM

```
<?xml version="1.0"?>
<launch>
    <arg name="sequence" default="sequence05"/>
    <arg name="speed" default="speed01"/>
    <arg name="record" default="false"/>


    <node pkg="rosbag" type="play" name="rosbag_play" args="--clock
     ↪  -r 1 -k
     ↪  /home/tor/Downloads/semantikitti2bag/semantickitti_$(arg
     ↪  sequence).bag" required="true" />

    <group if="$(arg record)">
    <node name = "rosbag_record" pkg="rosbag" type="record"
    args="/odom_pose /aft_mapped_to_init -O '/mnt/d/OneDrive -
     ↪  Norwegian University of Life Sciences/master/results/$(arg
     ↪  sequence)/scaloam_$(arg sequence)_$(arg speed).bag'"
    output="screen"/>
    </group>

    <include file="$(find
     ↪  sc_aloam_velodyne)/launch/aloam_velodyne_HDL_64.launch"/>

</launch>
```

Launch files for collected dataset

A-LOAM

```
<launch>
    <!-- Roslaunch /ouster/points topics to /velodyne_points -->
    <arg name="sequence" default="orchard"/>
    <arg name="speed" default="speed01"/>
    <arg name="record" default="false"/>
```

```
<node pkg="rosbag" type="play" name="rosbag_play" args="--clock -r
 ↪  1 '/mnt/e/master/data/$(arg sequence)_with_gps.bag'"
 ↪  required="true"/>

<param name="/use_sim_time" value="true" />

<group if="$(arg record)">
<node name = "rosbag_record" pkg="rosbag" type="record"
args="/gps_pose /aft_mapped_to_init -O '/mnt/d/OneDrive -
 ↪  Norwegian University of Life Sciences/master/results/$(arg
 ↪  sequence)/aloam_$(arg sequence)_$(arg speed).bag'"
output="screen"/>
</group>




<param name="scan_line" type="int" value="128" />

<!-- if 1, do mapping 10 Hz, if 2, do mapping 5 Hz. Suggest to use
 ↪  1, it will adjust frequence automaticlly -->
<param name="mapping_skip_frame" type="int" value="1" />

<!-- remove too closed points -->
<param name="minimum_range" type="double" value="5"/>


<param name="mapping_line_resolution" type="double" value="0.4"/>
<param name="mapping_plane_resolution" type="double" value="0.8"/>
<param name="point_cloud_topic" type="string"
 ↪  value="/ouster/points"/>

<node pkg="aloam_velodyne" type="ascanRegistration"
 ↪  name="ascanRegistration" output="screen" />
```

```
    <node pkg="aloam_velodyne" type="alaserOdometry"
     ↪  name="alaserOdometry" output="screen" />

    <node pkg="aloam_velodyne" type="alaserMapping"
     ↪  name="alaserMapping" output="screen" />

    <arg name="rviz" default="true" />
    <group if="$(arg rviz)">
        <node launch-prefix="nice" pkg="rviz" type="rviz" name="rviz"
         ↪  args="-d $(find
         ↪  aloam_velodyne)/rviz_cfg/aloam_velodyne.rviz" />
    </group>

</launch>
```

F-LOAM

```
<?xml version="1.0"?>
<launch>
    <arg name="sequence" default="orchard"/>
    <arg name="speed" default="speed01"/>
    <arg name="record" default="false"/>

    <node pkg="rosbag" type="play" name="rosbag_play" args="--clock -r
     ↪  1 '/mnt/e/master/data/$(arg sequence)_with_gps.bag'"
     ↪  required="true"/>


    <param name="/use_sim_time" value="true" />
    <group if="$(arg record)">
    <node name = "rosbag_record" pkg="rosbag" type="record"
    args="/gps_pose /odom -O '/mnt/d/OneDrive - Norwegian University
     ↪  of Life Sciences/master/results/$(arg sequence)/floam_$(arg
     ↪  sequence)_$(arg speed).bag'"
    output="screen"/>
    </group>
```

```
<!-- For Velodyne VLP-16
<param name="scan_line" value="16" />
-->

<!-- For Velodyne HDL-32
<param name="scan_line" value="32" />
-->

<!-- For Velodyne OS0-128 -->
<param name="point_cloud_topic" value="/ouster/points"/>
<param name="scan_line" value="128" />

<!--- Sim Time -->
<param name="/use_sim_time" value="true" />
<param name="scan_period" value="0.1" />

<param name="vertical_angle" type="double" value="2.0" />
<param name="max_dis" type="double" value="90.0" />
<param name="map_resolution" type="double" value="0.4" />
<param name="min_dis" type="double" value="4.0" />
<!--- -->
<node pkg="floam" type="floam_odom_estimation_node"
 ↪  name="floam_odom_estimation_node" output="screen"/>
<node pkg="floam" type="floam_laser_processing_node"
 ↪  name="floam_laser_processing_node" output="screen"/>
<node pkg="floam" type="floam_laser_mapping_node"
 ↪  name="floam_laser_mapping_node" output="screen"/>

<node pkg="tf" type="static_transform_publisher"
 ↪  name="word2map_tf"  args="0 0 0 0 0 0 /robot_odom /map 10" />

<arg name="rviz" default="true" />
<group if="$(arg rviz)">
    <node launch-prefix="nice" pkg="rviz" type="rviz" name="rviz"
     ↪  args="-d $(find floam)/rviz/floam_mapping.rviz" />
```

```
    </group>


        <node pkg="hector_trajectory_server"
        ↪  type="hector_trajectory_server"
        ↪  name="trajectory_server_loam" ns="gt" >
      <param name="/target_frame_name" value="world" />
      <param name="/source_frame_name" value="map" />
      <param name="/trajectory_update_rate" value="10.0" />
      <param name="/trajectory_publish_rate" value="10.0" />
    </node>
    <node pkg="hector_trajectory_server"
    ↪  type="hector_trajectory_server" name="trajectory_server_loam"
    ↪  ns="base_link" >
      <param name="/target_frame_name" value="world" />
      <param name="/source_frame_name"
      ↪  value="robot_top_3d_laser_link" />
      <param name="/trajectory_update_rate" value="10.0" />
      <param name="/trajectory_publish_rate" value="10.0" />
    </node>

</launch>
```

HDL-Graph SLAM

```
<?xml version="1.0"?>
<launch>
    <arg name="sequence" default="orchard"/>
    <arg name="speed" default="speed01"/>
    <arg name="record" default="false"/>

    <node pkg="rosbag" type="play" name="rosbag_play" args="--clock -r
    ↪  1 '/mnt/e/master/data/$(arg sequence)_with_gps.bag'"
    ↪  required="true"/>
    <param name="/use_sim_time" value="true" />
```

```
    <group if="$(arg record)">
    <node name = "rosbag_record" pkg="rosbag" type="record"
    args="/gps_pose /odom -O '/mnt/d/OneDrive - Norwegian University
     ↪  of Life Sciences/master/results/$(arg sequence)/hdl_$(arg
     ↪  sequence)_$(arg speed).bag'"
    output="screen"/>
    </group>


    <arg name="rviz" default="true" />
    <arg name="rviz_config" default="$(find
     ↪  hdl_graph_slam)/rviz/hdl_graph_slam.rviz" />

    <rosparam param="use_sim_time">true</rosparam>



    <include file="$(find
     ↪  hdl_graph_slam)/launch/hdl_graph_slam_robotnik.launch"/>

    <group if="$(arg rviz)">
        <node pkg="rviz" type="rviz" name="rviz" args="-d $(arg
         ↪  rviz_config)" />
    </group>

</launch>
```

KISS-ICP

```
<?xml version="1.0"?>
<launch>
    <arg name="sequence" default="orchard"/>
    <arg name="speed" default="speed01"/>
    <arg name="record" default="false"/>

    <!--
```

```
    <node pkg="rosbag" type="play" name="rosbag_play" args=" clock -r
     ↪  1 '/mnt/d/OneDrive - Norwegian University of Life
     ↪  Sciences/master/data/$(arg sequence)_with_gps.bag'"
     ↪  required="true"/>
    -->

    <node pkg="rosbag" type="play" name="rosbag_play" args="--clock -r
     ↪  1 '/mnt/e/master/data/$(arg sequence)_with_gps.bag'"
     ↪  required="true"/>

    <param name="/use_sim_time" value="true" />

    <group if="$(arg record)">
    <node name = "rosbag_record" pkg="rosbag" type="record"
    args="/gps_pose /kiss/odometry -O '/mnt/d/OneDrive - Norwegian
     ↪  University of Life Sciences/master/results/$(arg
     ↪  sequence)/kiss_icp_$(arg sequence)_$(arg speed).bag'"
    output="screen"/>
    </group>


    <include file="$(find kiss_icp)/launch/odometry.launch">
        <arg name="topic" value="ouster/points"/>
        <arg name="odom_frame" value="robot_odom"/>
    </include>


</launch>
```

LeGO-LOAM

```
<launch>
    <arg name="sequence" default="orchard"/>
    <arg name="speed" default="speed01"/>
    <arg name="record" default="false"/>
```

```
<node pkg="rosbag" type="play" name="rosbag_play" args="--clock -r
 ↪  1 '/mnt/e/master/data/$(arg sequence)_with_gps.bag'"
 ↪  required="true"/>


<group if="$(arg record)">
<node name = "rosbag_record" pkg="rosbag" type="record"
args="/gps_pose /aft_mapped_to_init -O '/mnt/d/OneDrive -
 ↪  Norwegian University of Life Sciences/master/results/$(arg
 ↪  sequence)/lego_$(arg sequence)_$(arg speed).bag'"
output="screen"/>
</group>

<!--- Sim Time -->
<param name="/use_sim_time" value="true" />

<!--- Run Rviz-->
<node pkg="rviz" type="rviz" name="rviz" args="-d $(find
 ↪  ouster_lego)/launch/test.rviz" />

<!--- TF -->
<node pkg="tf" type="static_transform_publisher"
 ↪  name="camera_init_to_map"  args="0 0 0 1.570795   0
 ↪  1.570795 /map    /camera_init 10" />
<node pkg="tf" type="static_transform_publisher"
 ↪  name="base_link_to_camera" args="0 0 0 -1.570795 -1.570795 0
 ↪  /camera /robot_base_link   10" />

<!--- LeGO-LOAM -->
<node pkg="ouster_lego" type="imageProjection"
 ↪  name="imageProjection"    output="screen"/>
<node pkg="ouster_lego" type="featureAssociation"
 ↪  name="featureAssociation" output="screen"/>
<node pkg="ouster_lego" type="mapOptmization"
 ↪  name="mapOptmization"     output="screen"/>
```

```
    <node pkg="ouster_lego" type="transformFusion"
     ↪  name="transformFusion"    output="screen"/>

</launch>
```

SC-A-LOAM

```
<?xml version="1.0"?>
<launch>
    <arg name="sequence" default="orchard"/>
    <arg name="speed" default="speed01"/>
    <arg name="record" default="false"/>

    <node pkg="rosbag" type="play" name="rosbag_play" args="--clock -r
     ↪  1 '/mnt/e/master/data/$(arg sequence)_with_gps.bag'"
     ↪  required="true"/>


    <group if="$(arg record)">
    <node name = "rosbag_record" pkg="rosbag" type="record"
    args="/gps_pose /aft_mapped_to_init -O '/mnt/d/OneDrive -
     ↪  Norwegian University of Life Sciences/master/results/$(arg
     ↪  sequence)/scaloam_$(arg sequence)_$(arg speed).bag'"
    output="screen"/>
    </group>


    <param name="scan_line" type="int" value="128" />

    <!-- if 1, do mapping 10 Hz, if 2, do mapping 5 Hz. Suggest to use
     ↪  1, it will adjust frequence automaticlly -->
    <param name="mapping_skip_frame" type="int" value="1" />

    <!-- remove too closed points -->
    <param name="minimum_range" type="double" value="5"/>
```

```
<param name="mapping_line_resolution" type="double" value="0.4"/>
 ↪  <!-- A-LOAM -->
<param name="mapping_plane_resolution" type="double" value="0.8"/>
 ↪  <!-- A-LOAM -->

<!-- SC-A-LOAM -->
<param name="keyframe_meter_gap" type="double" value="0.01"/>

<!-- Scan Context -->
<param name="sc_dist_thres" type="double" value="0.4"/> <!--
 ↪  SC-A-LOAM, if want no outliers, use 0.1-0.15 -->
<!-- <param name="sc_max_radius" type="double" value="40.0"/> 20
 ↪  or 40 for indoor -->
<param name="sc_max_radius" type="double" value="80.0"/> <!-- for
 ↪  outdoor -->

<!-- for MulRan -->
<param name="lidar_type" type="string" value="OS0-128"/>
<remap from="/velodyne_points" to="/ouster/points"/>

<!-- utils -->
<param name="save_directory" type="string"
 ↪  value="/home/tor/Documents/scloam/"/>  <!-- CHANGE THIS and
 ↪  end with / -->
<param name="point_cloud_topic" type="string"
 ↪  value="/ouster/points"/>

<!-- nodes -->
<node pkg="sc_aloam_velodyne" type="ascanRegistration"
 ↪  name="ascanRegistration" output="screen" /> <!-- A-LOAM -->
<node pkg="sc_aloam_velodyne" type="alaserOdometry"
 ↪  name="alaserOdometry" output="screen" /> <!-- A-LOAM -->
<node pkg="sc_aloam_velodyne" type="alaserMapping"
 ↪  name="alaserMapping" output="screen" /> <!-- A-LOAM -->
<node pkg="sc_aloam_velodyne" type="alaserPGO" name="alaserPGO"
 ↪  output="screen" /> <!-- SC-A-LOAM -->
```

```
    <!-- visulaization -->
    <arg name="rviz" default="true" />
    <group if="$(arg rviz)">
        <node launch-prefix="nice" pkg="rviz" type="rviz" name="rviz"
        ↪  args="-d $(find
        ↪  aloam_velodyne)/rviz_cfg/aloam_velodyne.rviz" />
    </group>


</launch>
```

Scripts

Covariance plotter

```
import rosbag
import matplotlib.pyplot as plt
from sensor_msgs.msg import NavSatFix

# Function to extract covariance from NavSatFix messages
def extract_covariance(bag):
    cov_lat = []
    cov_lon = []
    cov_alt = []
    times = []
    for topic, msg, timestamp in
    ↪  bag.read_messages(topics=['/robot/gps/fix']):
        cov_lat.append(msg.position_covariance[0])
        cov_lon.append(msg.position_covariance[4])
        cov_alt.append(msg.position_covariance[8])
        times.append(timestamp.to_sec())
    return cov_lat, cov_lon, cov_alt, times

# Load bag file
bag_path = "2024-04-10-14-22-54.bag"
bag = rosbag.Bag(bag_path)
```

```
# Extract covariance in latitude, longitude, and altitude over time
cov_lat, cov_lon, cov_alt, times = extract_covariance(bag)

# Plot covariance over time
plt.figure(figsize=(10, 6))
plt.plot(times, cov_lat, label='Covariance for Latitude')
plt.plot(times, cov_lon, label='Covariance for Longitude')
plt.plot(times, cov_alt, label='Covariance for Altitude')
plt.xlabel('Time (s)')
plt.ylabel('Covariance')
plt.title('Covariance in Latitude, Longitude, and Altitude Over Time')
plt.legend()
plt.grid(True)
plt.show()

# Close the bag
bag.close()
```

Gps to odom converter for lidar

```
#RUNS IN WSL, NO DOCKER, PYHTON 2
import rosbag
from geographiclib.geodesic import Geodesic
from geometry_msgs.msg import PoseStamped
from std_msgs.msg import Float64
import numpy as np
import tf
from collections import deque
from datetime import datetime, timedelta

def gps_to_local(lat, lon, alt, origin_lat, origin_lon, origin_alt):
    geod = Geodesic.WGS84
    g = geod.Inverse(origin_lat, origin_lon, lat, lon)
    forward_azimuth_rad = np.deg2rad(g['azi1'])
    x = g['s12'] * np.sin(forward_azimuth_rad)
```

```
    y = g['s12'] * np.cos(forward_azimuth_rad)
    z = alt - origin_alt
    return x, y, z



def process_rosbag(input_bag_path, output_bag_path):
    time_windows = []  # Time windows in seconds
    data_buffers = {window: deque() for window in time_windows}

    origin_set = False
    origin_lat = None
    origin_lon = None
    origin_alt = None

    with rosbag.Bag(output_bag_path, 'w') as outbag:
        for topic, msg, t in
         ↪  rosbag.Bag(input_bag_path).read_messages():
            if topic == '/robot/gps/fix':
                #print(msg.header.stamp.to_sec())
                if not origin_set:
                    origin_lat = msg.latitude
                    origin_lon = msg.longitude
                    origin_alt = msg.altitude
                    origin_set = True


                # Write the /odom_pose message for the latest received
                 ↪  point
                x, y, z = gps_to_local(msg.latitude, msg.longitude,
                 ↪  msg.altitude,
                                       origin_lat, origin_lon,
                                        ↪  origin_alt)
                odom_pose_msg = PoseStamped()
                odom_pose_msg.header.stamp = t
                odom_pose_msg.header.frame_id = 'robot_odom'
```

```
                odom_pose_msg.pose.position.x = x
                odom_pose_msg.pose.position.y = y
                odom_pose_msg.pose.position.z = z

                outbag.write('/gps_pose', odom_pose_msg, t)

            # Write the original topic and message to the bag
            outbag.write(topic, msg, t)

if __name__ == "__main__":
    input_bag_path = '../data/parking.bag'  # Replace with your input
    ↪ bag path
    output_bag_path = '../data/parking_with_gps.bag'  # Define your
    ↪ output bag path


    process_rosbag(input_bag_path, output_bag_path)
```

Pcd file calculator for volume and density

```
import os
import pandas as pd
import open3d as o3d

# Define the directory containing the PCD files
folder = "parking"
directory_path = 'maps/' + folder  # Adjust this to your folder path:w


# Initialize a DataFrame to store the results
results_df = pd.DataFrame(columns=[
    'File Name', 'File Size (MB)', 'Number of Points', 'Volume
    ↪ ($m^3$)',
    'Density (points/$m^3$)', 'Area ($m^2$)', 'Density
    ↪ (points/$m^2$)'])

# Iterate through all files in the directory
```

```
for filename in os.listdir(directory_path):
    if filename.endswith(".pcd"):
        # Load the PCD file
        file_path = os.path.join(directory_path, filename)
        pcd = o3d.io.read_point_cloud(file_path)

        # Get the file size in megabytes
        file_size = os.path.getsize(file_path) / 1048576  # Convert
        ↪ bytes to megabytes

        # Compute the axis-aligned bounding box
        aabb = pcd.get_axis_aligned_bounding_box()

        # Calculate the volume of the bounding box in scientific
        ↪ notation
        aabb_volume = aabb.volume()
        formatted_volume = f"{aabb_volume:.2e}"  # Format volume as
        ↪ scientific notation

        # Calculate the area of the base of the bounding box (x and y
        ↪ dimensions)
        aabb_extent = aabb.get_extent()
        base_area = aabb_extent[0] * aabb_extent[1]
        formatted_area = f"{base_area:.2e}"

        # Get the number of points
        num_points = len(pcd.points)

        # Calculate the density (points per unit volume)
        density_volume = num_points / aabb_volume if aabb_volume > 0
        ↪ else 0

        # Calculate the density (points per unit area)
        density_area = num_points / base_area if base_area > 0 else 0

        # Append results to DataFrame
```

```
        results_df.loc[len(results_df)] = [
            filename, round(file_size, 2), num_points,
            ↪  formatted_volume, density_volume, formatted_area,
            ↪  density_area]

# Convert DataFrame to LaTeX format
latex_output = results_df.to_latex(index=False, header=True,
↪  bold_rows=True, float_format="%.5f",
↪  label=folder+"ReconstrucedStats")

output_file_path = folder + 'ReconstrucedStats.tex'  # Specify your
↪  desired file path
# Save the LaTeX-formatted DataFrame to a .tex file
with open(output_file_path, 'w') as file:
    file.write("\\begin{table}[H]\n")
    file.write("\\centering\n")
    file.write(latex_output)
    file.write("\\caption{Reconstructed Statistics}\n")
    file.write("\\end{table}\n")

print(f"Saved LaTeX table to {output_file_path}")
```

ply to pcd converter

```
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 25 13:05:45 2024

@author: marko
"""

import open3d as o3d
pcd = o3d.io.read_point_cloud("RTABMAP_fr2_desk_reconstructed.ply")
o3d.io.write_point_cloud("RTABMAP_fr2_desk_reconstructed.pcd", pcd)
```

rosbag to image folder [72]

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Copyright 2021 ran.cheng2@mail.mcgill.ca

"""Extract images from a rosbag and synchronize the sequences
Please run this script with /usr/bin/python2.7 instead of python3
and don't forget to source the ros environment: source
 ↪  /opt/ros/melodic/setup.bash
"""

import os
import argparse
import numpy as np
import cv2
import copy
import rosbag
import shutil
from tqdm import tqdm
from sensor_msgs.msg import Image
from cv_bridge import CvBridge

def matching_time_indices(stamps_1, stamps_2,
                          max_diff = 0.1,
                          offset_2 = 0.0):
    """
    Searches for the best matching timestamps of two lists of
     ↪  timestamps
    and returns the list indices of the best matches.
    :param stamps_1: first vector of timestamps (numpy array)
    :param stamps_2: second vector of timestamps (numpy array)
    :param max_diff: max. allowed absolute time difference
    :param offset_2: optional time offset to be applied to stamps_2
```

```
    :return: 2 lists of the matching timestamp indices (stamps_1,
     ↪  stamps_2)
    """
    matching_indices_1 = []
    matching_indices_2 = []
    stamps_2 = copy.deepcopy(stamps_2)
    stamps_2 += offset_2
    for index_1, stamp_1 in enumerate(stamps_1):
        diffs = np.abs(stamps_2 - stamp_1)
        index_2 = int(np.argmin(diffs))
        if diffs[index_2] <= max_diff:
            matching_indices_1.append(index_1)
            matching_indices_2.append(index_2)
    return matching_indices_1, matching_indices_2

def main():
    """Extract a folder of images from a rosbag.
    """
    global seq_base_folder_sync
    parser = argparse.ArgumentParser(description="Extract images from
     ↪  a ROS bag.")
    parser.add_argument("bag_file", help="Input ROS bag.")
    parser.add_argument("output_dir", help="Output directory.")
    parser.add_argument("--tum", help="convert to tum mode.",
     ↪  dest="tum", type=bool, default=False)
    parser.add_argument("--image_topic", nargs='+', help="usage:
     ↪  --image_topic /image/data /depth/data")

    args = parser.parse_args()

    print "Extract images from %s on topic %s into %s" %
     ↪  (args.bag_file,

                                                    ↪  args.image_top
                                                    ↪  args.output_di
```

```
bag = rosbag.Bag(args.bag_file, "r")
bridge = CvBridge()
color_count = 0
depth_count = 0
color_stamps = {}
depth_stamps = {}  # use depth stamps as master and color as
 ↪  aligner for time stamp sync
if not os.path.exists(args.output_dir):
    os.mkdir(args.output_dir)
# you can use topics= [args.image_topic] to specify your topic
 ↪  name /device_0/sensor_0/Depth_0/image/data
 ↪  /device_0/sensor_1/Color_0/image/data
# for topic, msg, t in
 ↪  bag.read_messages(topics=["/camera/color/image_raw",
 ↪  "/camera/aligned_depth_to_color/image_raw"]):
if args.image_topic:
    img_topics = args.image_topic
else:
    img_topics = ["/camera/color/image_raw",
     ↪  "/camera/aligned_depth_to_color/image_raw"]
for topic, msg, t in bag.read_messages(
        topics=img_topics):
    cv_img = bridge.imgmsg_to_cv2(msg,
     ↪  desired_encoding="passthrough")
    output_fname = ""
    if not 'depth' in topic.lower() and 'color' in topic.lower():
        cv_img = cv2.cvtColor(cv_img, cv2.COLOR_BGR2RGB)
        color_directory_path = os.path.join(args.output_dir,
         ↪  "color")
        if not os.path.exists(color_directory_path):
            os.makedirs(color_directory_path)
        output_fname = os.path.join(color_directory_path,
         ↪  "%06i.png" % color_count)
        print "Wrote color image %i" % color_count
        color_stamps[msg.header.stamp.to_sec()] = output_fname
        color_count += 1
```

```
    if 'depth' in topic.lower():
        depth_directory_path = os.path.join(args.output_dir,
        ↪  "depth")
        if not os.path.exists(depth_directory_path):
            os.makedirs(depth_directory_path)
        output_fname = os.path.join(depth_directory_path,
        ↪  "%06i.png" % depth_count)
        print "Wrote depth image %i" % depth_count
        depth_stamps[msg.header.stamp.to_sec()] = output_fname
        depth_count += 1
    cv2.imwrite(output_fname, cv_img)
bag.close()
print("starting time sync...")
# start the time sync:
depth_stamps_t = np.fromiter(depth_stamps.iterkeys(), dtype=float)
color_stamps_t = np.fromiter(color_stamps.iterkeys(), dtype=float)
depth_stamps_t.sort()
color_stamps_t.sort()
# find the matching indices between depth and color time stamps
matching_indices_1, matching_indices_2 =
↪  matching_time_indices(depth_stamps_t, color_stamps_t)
# len(matching_indices_1) == len(matching_indices_2)
matched_depth_stamps_t = []
seq_base_folder_sync = args.output_dir + "_sync"
if args.tum:
    seq_base_folder_sync += "_tum"
seq_depth_folder_sync = os.path.join(seq_base_folder_sync,
↪  "depth")
if args.tum:
    seq_color_folder_sync = os.path.join(seq_base_folder_sync,
    ↪  "rgb")
else:
    seq_color_folder_sync = os.path.join(seq_base_folder_sync,
    ↪  "color")
if not os.path.exists(seq_base_folder_sync):
    os.makedirs(seq_base_folder_sync)
```

```
    os.makedirs(seq_depth_folder_sync)
    os.makedirs(seq_color_folder_sync)
for depth_iter_idx, dpt_index in
 ↪  tqdm(enumerate(matching_indices_1)):
    # obtain the file path of the matched indices
    depth_file_path = depth_stamps[depth_stamps_t[dpt_index]]
    color_file_path =
     ↪  color_stamps[color_stamps_t[matching_indices_2[depth_iter_idx]]]
    # save the depth and color sync files to the new folder with
     ↪  new indices
    if args.tum:
        # print "Converting to TUM dataset"
        shutil.copy(depth_file_path,
         ↪  os.path.join(seq_depth_folder_sync, "%1.6f.png" %
         ↪  depth_stamps_t[dpt_index]))
        shutil.copy(color_file_path,
         ↪  os.path.join(seq_color_folder_sync, "%1.6f.png" %
         ↪  depth_stamps_t[dpt_index]))
    else:
        # print "Converting to RealSense Dataset"
        shutil.copy(depth_file_path,
         ↪  os.path.join(seq_depth_folder_sync, "%06i.png" %
         ↪  depth_iter_idx))
        shutil.copy(color_file_path,
         ↪  os.path.join(seq_color_folder_sync, "%06i.png" %
         ↪  depth_iter_idx))
    # record the time stamp
    matched_depth_stamps_t.append([depth_iter_idx,
     ↪  depth_stamps_t[dpt_index]])
matched_depth_stamps_t = np.array(matched_depth_stamps_t)
print("end time sync.")
fmt = '%06d', '%1.6f'
np.savetxt(os.path.join(seq_base_folder_sync, "timestamps.txt"),
 ↪  matched_depth_stamps_t, fmt)
print("timestamps.txt saved to %s" %
 ↪  os.path.join(seq_base_folder_sync, "timestamps.txt"))
```

```
    return

if __name__ == '__main__':
    main()
```

From rosbag echo to foo.txt to GPS coordinates

```
input_file_path = '/home/vboxuser/Music/parking_gps.txt'
output_file_path = '/home/vboxuser/Music/parking_gps_1.txt'

def process_gps_data_final(file_path):
    with open(file_path, 'r') as file:
        data = file.read()

    entries = data.split('---')
    processed_data = []

    for entry in entries:
        lines = entry.strip().split('\n')
        gps_info = {}

        for line in lines:
            line = line.strip()
            if line.startswith('secs:'):
                gps_info['secs'] = line.split(':')[-1].strip()
            elif line.startswith('nsecs:'):
                gps_info['nsecs'] = line.split(':')[-1].strip()
            elif line.startswith('latitude:'):
                gps_info['latitude'] = line.split(':')[-1].strip()
            elif line.startswith('longitude:'):
                gps_info['longitude'] = line.split(':')[-1].strip()
            elif line.startswith('altitude:'):
                gps_info['altitude'] = line.split(':')[-1].strip()

        if all(key in gps_info for key in ['secs', 'nsecs',
        ↪  'latitude', 'longitude', 'altitude']):
            timestamp = f"{gps_info['secs']}.{gps_info['nsecs']}"
```

```
            processed_line = f"{timestamp} {gps_info['latitude']}
             ↪  {gps_info['longitude']} {gps_info['altitude']}"
            processed_data.append(processed_line)

    return processed_data

def save_processed_data(processed_data, output_file_path):
    with open(output_file_path, 'w') as file:
        for line in processed_data:
            file.write(line + '\n')

processed_gps_data_final = process_gps_data_final(input_file_path)

save_processed_data(processed_gps_data_final, output_file_path)
```

GPS to XYZ converter for camera based SLAM systems

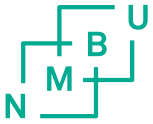
```
import numpy as np
import pandas as pd

file_path = '/home/vboxuser/Music/parking_gps_1.txt'
gps_data = pd.read_csv(file_path, sep=" ", header=None,
 ↪  names=["timestamp", "latitude", "longitude", "altitude"])

R = 6371000  # Earth's radius in meters

# Convert degrees to radians
gps_data['latitude_rad'] = np.radians(gps_data['latitude'])
gps_data['longitude_rad'] = np.radians(gps_data['longitude'])

# Convert from GPS coordinates to Cartesian coordinates
gps_data['x'] = (R + gps_data['altitude']) *
 ↪  np.cos(gps_data['latitude_rad']) *
 ↪  np.cos(gps_data['longitude_rad'])
gps_data['y'] = (R + gps_data['altitude']) *
 ↪  np.cos(gps_data['latitude_rad']) *
 ↪  np.sin(gps_data['longitude_rad'])
```

```
gps_data['z'] = (R + gps_data['altitude']) *
 ↪  np.sin(gps_data['latitude_rad'])

# Quaternion values to append
quaternion_str = "0.000000 0.000000 0.000000 1.000000"

formatted_lines = gps_data.apply(lambda row: f"{row['timestamp']}
 ↪  {row['x']} {row['y']} {row['z']} {quaternion_str}", axis=1)

# Save the formatted lines to a new TXT file
output_file_path = '/home/vboxuser/Music/orchard_groundtruth.txt'
with open(output_file_path, 'w') as file:
    for line in formatted_lines:
        file.write(f"{line}\n")

print(output_file_path)
```

Creating rgb.txt and depth.txt from rgb/depth folder

```
import os
import sys

def process_images(folder_path, output_file):
    if not os.path.isdir(folder_path):
        print(f"The directory {folder_path} does not exist.")
        sys.exit(1)

    files = [f for f in os.listdir(folder_path) if f.endswith('.png')]
    sorted_files = sorted(files, key=lambda x: float(x.rsplit('.',
     ↪  1)[0]))

    with open(output_file, 'w') as file:
        for file_name in sorted_files:
            timestamp = os.path.splitext(file_name)[0]
            file.write(f"{timestamp}
             ↪  {os.path.basename(folder_path)}/{file_name}\n")
```

```python
if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Usage: python process_images.py <folder_path>
        ↪  <output_file>")
        sys.exit(1)

    folder_path = sys.argv[1]
    output_file = sys.argv[2]
    process_images(folder_path, output_file)
```



Norges miljø- og biovitenskapelige universitet
Noregs miljø- og biovitskapelege universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway