Technische Universität München
TUM School of Computation, Information and Technology

3D LiDAR Odometry and Mapping Leveraging Prior Knowledge

Martin Oelsch, M.Sc.

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Die Dissertation wurde am 17.06.2022 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 17.11.2022 angenommen.

Abstract

In recent years, autonomous robots are increasingly used to conduct maintenance or inspection tasks. Infrastructure, such as bridges, buildings, windmills, and also airplanes need to be visually inspected regularly. Rifts in the facade of buildings and impacts of lightning strikes on airplanes can pose a severe safety risk.

The development of inexpensive Unmanned Aerial Vehicles (UAVs) has recently been the focus of many researchers and engineers. When a UAV is used for visual inspection, a very high localization and control precision are required to take close-up images while avoiding a collision. However, GPS is often unreliable close to buildings or indoors. Also, algorithms for Simultaneous Localization and Mapping (SLAM) are subject to drift, especially in long-term exploration. In this regard, prior knowledge about a reference object can improve SLAM accuracy.

To this end, this thesis proposes methods to further improve 3D LiDAR-SLAM accuracy by leveraging prior knowledge:

The **first contribution** presents an approach to building an initial map from the starting location of the robotic platform. The initial map is created by accumulating scans from an actuated LiDAR sensor. The key contribution is an approach to leave the voxelized feature maps of the initial map unchanged (immutable) while dynamically extending the maps during the exploration phase. Experiments show that leaving the initial map unchanged in a static environment improves the localization accuracy and map quality.

The **second contribution** leverages prior knowledge about a 3D reference object to improve 3D LiDAR-SLAM accuracy. Here, the location of the reference object in a global coordinate frame and its geometry in form of a 3D triangular mesh are considered prior knowledge. The approach allows the direct use of the triangular mesh, instead of requiring a prior conversion to a point cloud. In a novel formulation, the reference object is tightly coupled in a joint map optimization.

The **third contribution** proposes an approach for reference object-based trajectory and map optimization. This method loosely couples the 3D model of the reference object into the LiDAR-SLAM algorithm, allowing for a remote SLAM setup. Whenever a scan can be aligned well to the surface of the reference object, an optimization of the past trajectory is performed. In addition, drift in the map is corrected, improving the map quality for future localization. Due to the loose coupling of the reference object's 3D model, the proposed method is suitable for Edge Cloud processing.

Kurzfassung

In den letzten Jahren werden zunehmend autonome Roboter eingesetzt, um Wartungs- oder Inspektionsaufgaben durchzuführen. Infrastruktur wie Brücken, Gebäude, Windmühlen und auch Flugzeuge müssen regelmäßig einer Sichtprüfung unterzogen werden, da Risse in der Fassade von Gebäuden und Blitzeinschläge in Flugzeuge ein erhebliches Sicherheitsrisiko darstellen können.

Die Entwicklung kostengünstiger UAVs stand in letzter Zeit im Fokus von Forschern und Ingenieuren. Bei der visuellen Inspektion mit einem UAV ist eine sehr hohe Lokalisierungs- und Steuerungspräzision erforderlich, um Nahaufnahmen zu machen und gleichzeitig eine Kollision zu vermeiden. In der Nähe von Gebäuden oder in Innenräumen ist GPS jedoch oft unzuverlässig. Algorithmen für Simultaneous Localization and Mapping (SLAM) unterliegen ebenso Abweichungen, insbesondere bei langfristiger Exploration. In dieser Hinsicht kann Vorwissen über ein Referenzobjekt die SLAM-Genauigkeit verbessern.

Zu diesem Zweck schlägt diese Arbeit Methoden vor, um die Genauigkeit von 3D LiDAR-SLAM durch die Nutzung von Vorwissen weiter zu verbessern:

Der **erste Beitrag** stellt einen Ansatz vor, um eine anfängliche Karte vom Startort der Roboterplattform aus zu erstellen. Die anfängliche Karte wird durch Sammeln von Scans von einem aktuierten LiDAR-Sensor erstellt. Der Schlüsselbeitrag ist ein Ansatz, die voxelisierten Merkmalskarten der anfänglichen Karte unverändert zu lassen, während die Karten während der Erkundungsphase dynamisch erweitert werden. Experimente zeigen, dass das unveränderte Belassen der ursprünglichen Karte in einer statischen Umgebung die Lokalisierungsgenauigkeit und die Kartenqualität verbessert.

Der **zweite Beitrag** nutzt Vorwissen über ein 3D-Referenzobjekt, um die Genauigkeit von 3D LiDAR-SLAM zu verbessern. Dabei werden die Lage des Bezugsobjekts in einem globalen Koordinatensystem und seine Geometrie in Form eines 3D-Dreiecksnetzes als Vorwissen betrachtet. Der Ansatz ermöglicht die direkte Verwendung des Dreiecksnetzes, anstatt eine vorherige Umwandlung in eine Punktwolke zu erfordern. In einer neuartigen Formulierung wird das Referenzobjekt mit in die Kartenoptimierung eingebunden.

Der **dritte Beitrag** schlägt einen Ansatz zur referenzobjektbasierten Trajektorien- und Kartenoptimierung vor. Diese Methode koppelt das 3D-Modell des Referenzobjekts lose in den LiDAR-SLAM-Algorithmus, was einen Einsatz als Remote-SLAM ermöglicht. Immer wenn ein Scan gut an die Oberfläche des Referenzobjekts angelegt werden kann, wird eine Optimierung der vergangenen Trajektorie durchgeführt. Darüber hinaus wird der Drift in der Karte korrigiert, wodurch die Kartenqualität für die zukünftige Lokalisierung verbes-

sert wird. Aufgrund der losen Kopplung des 3D-Modells des Referenzobjekts eignet sich das vorgeschlagene Verfahren für Edge-Cloud-Verarbeitung.

Acknowledgments

I am especially grateful to my doctoral advisor Prof. Dr.-Ing. Eckehard Steinbach, who gave me the opportunity at the Chair of Media Technology back in 2016. I never regretted any minute of the six years, despite the usual hiccups one might experience. I will certainly remember your kindness, professional diplomacy, and excellent project management skills.

I would also like to thank Prof. Dr.-Ing. Thomas Eibert and Prof. Dr.-Ing. habil. Gerhard Rigoll, who accepted to be part of the examination committee and review this dissertation. Further, I would like to thank Prof. Dr. rer. nat. Bernhard Glavina, who agreed to be the mentor after supervising my Bachelor's and Master's theses.

Furthermore, I want to thank my former office colleagues from the VaMEx-VIPE project Dr.-Ing. Dominik Van Opdenbosch, Adrian Garcea and Dr.-Ing. Dmytro Bobkov. You picked me up back in 2016, and I learned so much from you over all these years. Also special thanks to the IT competence center around Martin Piccolrovazzi, and former office colleagues Michael Adam and Sebastian Eger. One thing I will certainly miss is firing up the servers with my endless computations. Unforgotten is also the help from Michael Adam when preparing me for the IVC lecture at Tongji University.

Generally, I am thankful for the patience of my colleagues at the Chair of Media Technology in the canteen when they had to wait for me as a slow eater. Ultimately, I think I mostly managed the 12:15 deadline.

My special gratitude goes to my AI-Inspection Drone project colleague Dr.-Ing. Mojtaba Leox Karimi. Our teamwork in these four years has been outstanding, and we managed to build a functional autonomous drone system for airplane surface inspection. Without your electrical and crafting talent, this would not have been possible. You are a true handyman. I will never forget our numerous trips to the sports hall and our long drives to Hamburg or even the flight to the drone workshop in Greece. In that regard, I would also like to express a big thanks to Dr.-Ing. Martin Maier for his efforts and contacts to organize our drone testing trips to the sports hall and turbine hall.

Ultimately, I want to thank my parents, Simone and Andreas, and my wife Lucia, for their endless support during my studies and these past six years. Thank you so much!

Martin Oelsch *Munich, March, 2023*

Contents

Notation

Abbreviations

Abbreviation	Description	Definition
AABB	Axis-aligned Bounding Box	page 24
APE	Absolute Pose Error	page 33
AR	Augmented Reality	page 7
ATE	Absolute Trajectory Error	page 33
BIM	Building Information Modeling	page 3
BVH	Bounding Volume Hierarchy	page 24
CAD	Computer-aided Design	page 2
CGAL	Computational Geometry Algorithms Library	page 15
CNN	Convolutional Neural Network	page 38
DoF	Degree-of-Freedom	page 7
EKF	Extended Kalman Filter	page 2
FCGF	Fully Convolutional Geometric Feature	page 17
FPFH	Fast Point Feature Histogram	page 17
GUI	Graphical User Interface	page 13
GT	Ground-truth	page 28
hFoV	Horizontal Field-of-View	page 1
ICF	Iterative Closest Face	page 45
ICP	Iterative Closest Point	page 17
IMU	Inertial Measurement Unit	page 2
KF	Kalman Filter	page 35
LiDAR	Light Detection and Ranging	page 1
LOAM	LiDAR Odometry and Mapping	page 3
LRF	Laser Range Finder	page 46
MAV	Micro Aerial Vehicle	page 17
MEMS	Micro-Electro-Mechanical System	page 10
MSE	Mean Squared Error	page 22
OBB	Oriented Bounding Box	page 24
PCA	Principal Component Analysis	page 45
PCD	Point Cloud Data file format	page 13
PCL	Point Cloud Library	page 13
PFH	Point Feature Histograms	page 17
PGO	Pose Graph Optimization	page 4
PSO	Particle Swarm Optimization	page 45
RE	Rotational Error	page 33
R-LOAM	Reference-LOAM	page 75
RMSE	Root Mean Squared Error	page 34
RO-LOAM	Reference Object-LOAM	page 96
ROS	Robot Operating System	page 38

Abbreviation	Description	Definition
SLAM	Simultaneous Localization And Mapping	page 2
SVD	Singular Value Decomposition	page 31
TMO	Trajectory and Map Optimization	page 4
UAV	Unmanned Aerial Vehicle	page 2
vFoV	Vertical Field-of-View	page 1
VR	Virtual Reality	page 7

Scalars, vectors and matrices

x	scalar
$\boldsymbol{x}$	vector
$\boldsymbol{X}$	matrix
$\lvert x \rvert$	absolute value of scalar x
$\lfloor x \rfloor$	floor operation on the scalar x
$\lVert \boldsymbol{x} \rVert_2$	euclidean norm (L2-norm) of vector $\boldsymbol{x}$
A	point A in the context of 3D geometry
$\boldsymbol{AB}$	vector from point A to B
$\boldsymbol{AB} \cdot \boldsymbol{BC}$	dot product between two vectors
$\boldsymbol{AB} \times \boldsymbol{BC}$	cross product between two vectors
$\mathcal{P}$	single point cloud
p	single point p, or as part of a point cloud $\mathcal{P}$
$\overrightarrow{\mathcal{P}}$	vector or list of point clouds $\mathcal{P}$
$\mathcal{V}$	set of vertices as part of a mesh
$\mathcal{F}$	set of faces as part of a mesh
$\mathcal{G}$	pose graph
$\boldsymbol{\Omega}$	6×6 information matrix
c	single correspondence, e.g., point-to-point or point-to-model
C	set of correspondences
ϕ	roll (Euler angle), or elevation angle
θ	pitch (Euler angle)
ψ	yaw (Euler angle)
ρ	loss function
$\mathcal{T}$	k-d tree
$\boldsymbol{t}$	translational vector
$\boldsymbol{q}$	quaternion
$\boldsymbol{R}$	3×3 rotation matrix

Subscripts and superscripts

$\bar{x}$	mean/centroid of x
$\hat{x}$	estimated/predicted/distorted value of x
$\bar{\bar{X}}$	cardinality (size) of the set X
p^{L_t}	point p in the frame $\{L\}$ recorded at time t
$\mathcal{P}^{L_t}$	point cloud $\mathcal{P}$ in the frame $\{L\}$ recorded at time t
$\mathcal{P}_s$	source point cloud for registration
$\mathcal{P}_t$	target point cloud for registration
$\mathcal{P}_\mathcal{E}$	point cloud of corner features
$\mathcal{P}_\mathcal{H}$	point cloud of surface features
$\mathcal{P}_\mathcal{V}$	point cloud of virtual points on a mesh surface
$C_\mathcal{E}$	set of corner point-to-point correspondences
$C_\mathcal{H}$	set of surface point-to-point correspondences
$C_\mathcal{M}$	set of point-to-mesh or point-to-model correspondences
${}^{W}\boldsymbol{T}_L$	homogeneous 4×4 transformation matrix of the frame $\{L\}$ in frame $\{W\}$
$\boldsymbol{T}^{-1}$	inverse of the transformation matrix $\boldsymbol{T}$
$\boldsymbol{T}^T$	transpose of the transformation matrix $\boldsymbol{T}$
$\boldsymbol{T}_{\text{SLAM},1}$	first pose of a SLAM trajectory
$\boldsymbol{T}_{\text{SLAM},1:n}$	poses 1 to n of a SLAM trajectory
$\hat{\boldsymbol{T}}$	projected/predicted transformation matrix
$\tilde{\boldsymbol{T}}$	refined transformation matrix $\boldsymbol{T}$ after model alignment
$\bar{\boldsymbol{T}}$	refined transformation matrix $\boldsymbol{T}$ after Pose Graph Optimization (PGO)

Chapter 1

Introduction

1.1 Motivation

Robotic platforms in outdoor scenarios or autonomous cars may determine their position with GPS, 5G, or even WiFi signals. The challenge is to acquire an as-high-as-possible update rate from sensors to frequently capture the state of the environment, while at the same time ensuring real-time processing of the recorded data. With 5G, some of the data may be offloaded to an Edge Cloud with high processing performance, while ensuring low latency. Autonomous cars mainly rely on GPS for localization and only use visual data for traffic analysis, i.e., traffic light or sign detection and road safety. For the safety of pedestrians and other road users, autonomous cars are equipped with several cameras and/or Light Detection and Ranging (LiDAR) sensors.

LiDAR sensors exist in various price ranges, accuracies, and properties. Some of the first were 1D range-finders, returning only the distance of a single point. Later on, 2D LiDAR sensors were developed, returning several measurements distributed along the horizontal Field-of-View (hFoV) of a single scan line. The FoV may be constrained or even reach full $360°$. Most recently, 3D LiDAR sensors are widely used for robotic applications. They have several scan lines distributed along a vertical Field-of-View (vFoV). The current trend increases the number of scan lines and therefore the vertical point density, but also the maximum measurement range. Typically, LiDAR sensors have a range of a few meters up to several hundred meters with ranging errors of only a few centimeters. LiDAR sensors have the advantage of being independent of light, i.e., night or bright daylight should not influence the measurements. Newest advances in LiDAR technology aim at increasing the reliability of measurements even for difficult visibility conditions, e.g., rain, snow, and fog. However, cameras have the advantage of higher resolution, higher update rates, and low cost. For this reason, cameras and LiDAR sensors are often combined for autonomous navigation or other robotic tasks.

Apart from automotive purposes, LiDAR sensors and cameras are often mounted on robotic platforms for autonomous tasks, e.g., package delivery service or assembly tasks with manipulators. Robotic platforms, however, often operate in indoor environments and require highly accurate positioning. LiDAR sensors may also be used to generate frequent, highly accurate digital twins in an indoor or outdoor environment. Digital twins are reconstructions in form of a point cloud, typically in 3D, that can be used as input to point cloud segmentation and object classification algorithms for inventory applications or process analysis. The digital twin can then be further processed and converted to a mesh representation, which may have a better visual appearance and reduced file size. Digital twins require a frequent update in dynamic indoor environments and may be created by mobile or static sensors. Mobile mapping backpacks or trolleys, for example, use Simultaneous Localization and Mapping (SLAM) algorithms for digital twin creation. In contrast, static sensors may be more accurate, but require an extrinsic calibration and are fixed to the room.

SLAM is still one of the major challenges for autonomous robots, which can not make use of accurate global positioning systems, e.g., GPS. SLAM is often referred to as a "chicken-or-egg" problem since the robot needs to localize itself within the map while at the same time extending the map. SLAM algorithms exist for a variety of sensors, such as cameras or LiDARs. Sometimes the computed poses are fused as odometry input in an Extended Kalman Filter (EKF) with measurements of Inertial Measurement Units (IMUs) to further improve localization accuracy and mapping quality, e.g., visual-inertial SLAM. If the robot starts without prior knowledge about the environment, the map is empty initially. While sensor data is captured and the robot explores the environment, a map is built and the position is continuously estimated. The map may not only be used for self-localization, but also for collision avoidance and path planning. In controlled environments, a highly accurate map as a digital twin can be built prior to the robotic exploration in an offline process. It can then be used by the SLAM algorithm as an initial map, i.e., prior knowledge about the environment to improve localization accuracy. The initial map can be as accurate as a digital twin or just be a representation of the room outline, e.g., an emergency floor plan in the form of a 2D or 3D point cloud.

If an initial map is given to the robot, it first has to localize itself within the map. This is known as the "kidnapped robot" problem. An initial guess about the position can result in a more accurate localization. Rather than creating an extensive prior map, only parts of the environment can be mapped and then be extended by the robot when exploring areas that were not previously visited. The map may also be updated if some parts have changed. Prior knowledge does not only have to be a previously built map but can also just be about the geometry and location of a known object in the environment. This enables the robot not only to localize within the map but also relative to the known object. Effectively, this enables another source for reference. As long as the robot stays in the vicinity of the reference object, it can make use of the absolute position within the map, but it can also relatively localize to the reference object.

In recent years, there is a growing demand for services providing visual inspection of infrastructure using Unmanned Aerial Vehicles (UAVs). Airplanes, train rails, windmills, bridges, or even large buildings need to be inspected on a regular basis to identify potential damages, which might pose a safety risk in the future. At the same time, architectural Computer-aided Design (CAD) models of the objects become available from manufacturers or highly accurate 3D models can be generated using devices from Building Information Modeling (BIM). 3D triangular meshes can be converted to point clouds or vice versa.

The generated 3D models can then be used as prior knowledge to improve localization accuracy and map quality. They can be applied to all inspection objects of the same type, or, can be reused for future inspections at the same site. Methods proposed in this thesis leverage this knowledge about the 3D geometry, while at the same time not making any assumptions about the surroundings. For example, this allows for airplane inspections in any kind of hangar or even outdoors, as long as the airplane type is the same.

Autonomous inspections require a highly accurate localization and control algorithm. Usually, high-resolution images of the inspection object's surface are taken and either transferred wirelessly or stored onboard for offline analysis. Lightning strikes on airplanes can have the size of a millimeter and pose a safety risk. To detect these damages, an autonomous inspection UAV needs to take close-up images. This also holds for many other inspection tasks. Therefore, highly accurate localization is required to pinpoint potential damage on the object's surface after its identification on the inspection images.

1.2 Major contributions

The goal of this thesis is to present novel methods to improve 3D LiDAR-SLAM accuracy with prior knowledge about the environment. Prior knowledge in this thesis is defined in two ways: The first way leverages an initial map, which was created by the robot at its starting location and is used as prior knowledge for the exploration. Assuming a static environment, it is argued that by keeping the initial map unchanged during the exploration, a further improvement in LiDAR-SLAM accuracy can be achieved. The second way leverages the knowledge about the position, orientation, and geometry of a 3D reference object.

The major contributions of this thesis are based on the LiDAR Odometry and Mapping framework (LOAM) [12] and are as follows:

1. A modification of the LOAM framework is proposed to enable **immutable initial map creation**. An initial map is built at the starting location by the robot itself by rotating a 3D LiDAR sensor with an actuator. A highly accurate map is created by accumulating LiDAR scans in a common coordinate frame with transforms reported by the actuator. The immutable initial map is extended throughout the exploration while leaving the points of the initial map unchanged. The basic idea is that the initial map is created with high accuracy while the robot is still static and points acquired during exploration are added with transforms from the estimated motion and are subject to higher insertion errors. It is shown that an immutable (fixed) initial map yields superior localization performance in a static environment compared to using a mutable (variable) initial map or conventional LOAM, which starts with an empty map.

2. A **joint map-optimization formulation** for the LOAM algorithm [12] is proposed, which not only makes use of conventional point-to-point correspondences between the LiDAR scan and the map but also takes point-to-mesh correspondences between

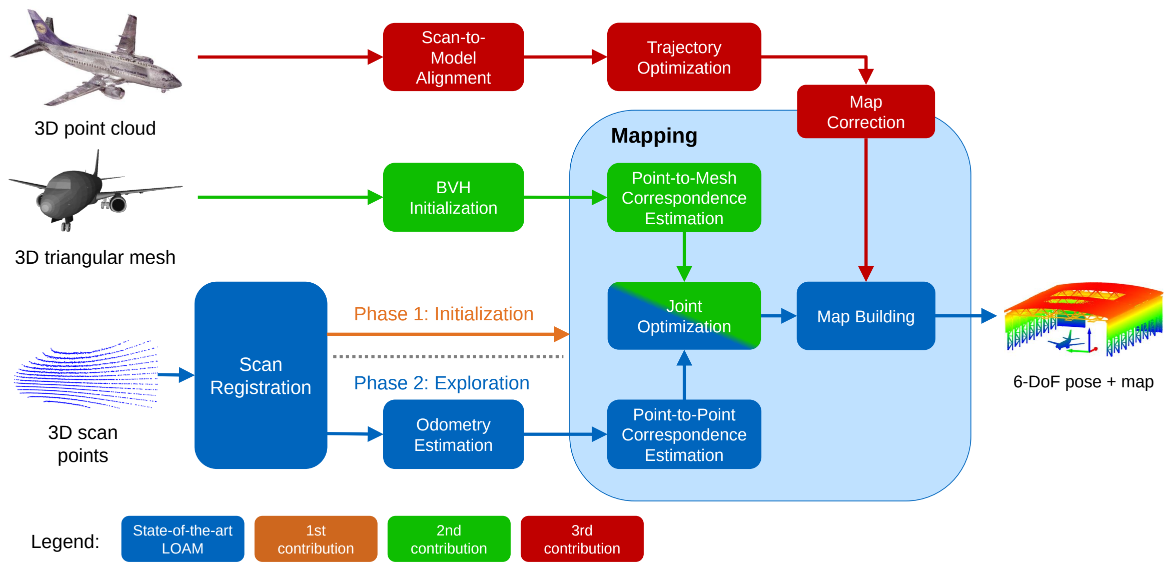


Figure 1.1: Integration overview of the contributions to the state-of-the-art LOAM framework [12]. Modifications are marked in different colors according to their contribution.

the LiDAR scan and a 3D reference object into account. The proposed method tightly couples the 3D mesh of the reference object to the map optimization process of conventional LOAM. The prior knowledge is seamlessly integrated and considered for each processed LiDAR scan. Whenever the robot is in the vicinity of the reference object, point-to-mesh correspondences are additionally added to the map optimization, increasing localization accuracy.

3. A **loosely coupled reference object-based trajectory and map optimization** method is proposed. LiDAR scans are aligned to the 3D reference object with map-optimized poses as initial guesses, yielding highly accurate model-converged poses as candidates for Trajectory and Map Optimization (TMO). The candidates are then verified by a motion prior check of an EKF. If the verification is successful, a partial Pose Graph Optimization (PGO) is triggered, correcting the poses of the conventional LOAM algorithm [12] since the previous TMO. A map correction module rebuilds the map since the last TMO with the corrected poses. The proposed approach reduces drift and improves long-term map quality. In addition, it allows for frequent TMOs with high accuracy, without the need for revisiting a place or an initial map.

Figure 1.1 gives an overview of the contributions in this thesis and how they are integrated into the existing LOAM framework.

The **first** contribution adds an initialization phase, creating an immutable initial map by accumulating scans at the starting location of the robot. The **second** contribution tightly couples a 3D triangular mesh as prior knowledge into the map optimization process of LOAM. The known 3D reference object effectively guides the SLAM process with its global cues. The **third** contribution leverages a known 3D point cloud model by finding highly accurate absolute poses with scan-to-model alignments. These absolute poses are then used to perform trajectory and map optimization to improve future localization and mapping. It can be seen that the third contribution is loosely coupled in comparison to the second contribution. Hence, this makes it especially suitable for a remote SLAM setup, e.g., offloaded processing to an Edge Cloud.

1.3 Thesis structure

The thesis is structured as follows:

Chapter 2 elaborates on the background of LiDAR sensors, point cloud acquisition, representations, and registration. Furthermore, different methods for trajectory alignment are explained in detail and evaluation error metrics are discussed. The EKF as state estimator finalizes the background section. In the related work section, state-of-the-art LiDAR-SLAM algorithms are discussed and the well-known LOAM algorithm [12] is explained in detail. Also, related works leveraging prior knowledge, e.g., by using prior maps, CAD models, or emergency floor plans are discussed. Finally, publications in the area of trajectory and map optimization by closing loops or relocalization are elaborated. Chapter 3 explains the first contribution in detail by creating an immutable initial map to improve LOAM accuracy. Chapter 4 presents the second contribution, by tightly coupling a known 3D triangular mesh into the map optimization process of LOAM. Chapter 5 presents the third contribution, by improving LOAM with scan-to-model alignments with a known 3D reference object and subsequent trajectory and map optimization. This thesis is concluded with a summary of the proposed methods and possible future improvements in Chapter 6.

Parts of this thesis have been published in peer-reviewed conferences [5] and journals [1], [2]. Other works outside the scope of this thesis were published in [6], [7], [3], [4], [8]–[11].

Chapter 2

Background and related work

This chapter first gives explanations and details about the background of LiDAR-SLAM, ranging from the acquisition of 3D points using LiDAR sensors to the detailed explanation of the LOAM [12] algorithm. The second section elaborates on related work in the area of LiDAR-SLAM, leveraging prior knowledge and trajectory and map optimization.

2.1 Background

2.1.1 LiDAR-based Simultaneous Localization and Mapping (LiDAR-SLAM)

SLAM has various applications ranging from autonomous robots and cars to Virtual Reality (VR) and Augmented Reality (AR). Outdoors, global positioning, and navigation systems such as GPS can be used to retrieve the current position in a global coordinate frame. In some scenarios, however, global positioning may not be available or is too imprecise for specific applications. Indoors, for example, the current position can be determined by prior knowledge about the environment, i.e., extrinsically calibrated wireless hotspots or a previously built map for visual or LiDAR-based localization. If no prior knowledge exists, a full SLAM algorithm needs to be employed, building a new map from scratch. An autonomous delivery UAV may rely on GPS data to find its way to the delivery location. Once arriving, local sensors on the UAV need to record data of the surroundings to identify suitable drop-off locations with segmentation and classification algorithms. A SLAM algorithm is then required to navigate the UAV for an exact parcel drop-off, but also for collision avoidance. Due to its robustness to lighting and weather conditions, LiDAR sensors are often used to perform SLAM.

LiDAR-SLAM is the challenge of creating a map of the environment, which is dependent on the correct insertion of new scans by self-localization within the map. It is sometimes referred to as a "chicken-or-egg" problem since localization and mapping are both dependent on each other.

In SLAM terminology, a pose in 3D space is consisting of the position with x, y, and z coordinates, as well as the orientation $roll$, $pitch$, and yaw. It is therefore considered as a 6-Degree-of-Freedom (DoF) pose. Often, the orientation is converted and processed as quaternion rather than Euler angles for computational purposes. The position can be written as a translational vector $\boldsymbol{t} = [t_x, t_y, t_z]$. The complete 6-DoF pose can be expressed as a 4×4 homogeneous transformation matrix:

$$\boldsymbol{T} = \begin{bmatrix} R_{1,1} & R_{1,2} & R_{1,3} & t_x \\ R_{2,1} & R_{2,2} & R_{2,3} & t_y \\ R_{3,1} & R_{3,2} & R_{3,3} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

or in short

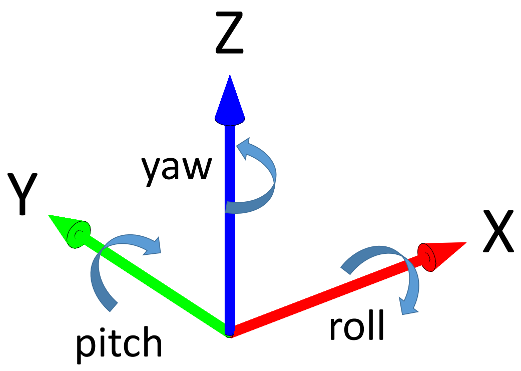
$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ 0 & 1 \end{bmatrix} \tag{2.2}$$

$\boldsymbol{R}$ is the 3×3 rotation matrix, describing the orientation, and $\boldsymbol{t}$ is the translational vector, describing the position of the pose.

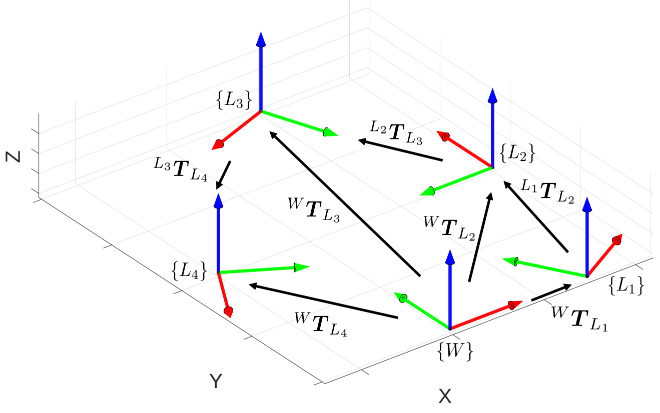
Since a 6-DoF *pose* can be directly described as a *transformation matrix* $\boldsymbol{T}$, both terms are used interchangeably in this thesis. A pose $\boldsymbol{T}$ and an acquired LiDAR point cloud $\mathcal{P}$ are always relative to a coordinate frame. Newly captured LiDAR data is in the sensor frame by default, denoted as $\{L\}$ in this thesis. The map coordinate frame, sometimes referred to as world $\{W\}$, is initialized when capturing the first LiDAR scan. Hence, the first pose is set to $\boldsymbol{T} = \boldsymbol{I}$, also called the *origin*. Once the robot starts exploring the environment, the pose of the LiDAR sensor in the map frame at a time t can be described with ${}^{W}\boldsymbol{T}_{L_t}$. The point cloud of the LiDAR scan from which the pose was computed is then denoted as $\mathcal{P}^{L_t}$. Essentially, it is the task of the SLAM algorithm to compute ${}^{W}\boldsymbol{T}_{L_t}$ from $\mathcal{P}^{L_t}$. The LiDAR scan can then be transformed to the map frame $\mathcal{P}^{W_t}$ for insertion. Usually, the SLAM algorithm estimates the pose with extracted point features. These are then inserted into the map, instead of the LiDAR points directly.

The full trajectory is the sequence of individual poses. The coordinate frames for LiDAR-SLAM systems and also other robotic applications follow the right-hand rule for the axis orientation. When mimicking the coordinate axes illustrated in Figure 2.1(a) with the first three fingers of the right hand, the index finger will point to the X-axis, the middle finger to the positive Y direction, and the thumb upwards along the Z-axis. Positive rotations for $roll$, $pitch$, and yaw follow the directions of the arrows as illustrated in the figure. Figure 2.1(b) shows a sequence of four poses $\{L_1\}$ to $\{L_4\}$, e.g., of a moved LiDAR sensor, and the world coordinate frame $\{W\}$ with $\boldsymbol{T} = \boldsymbol{I}$ representing the origin. Typically, the trajectory of a robotic system is evaluated with all poses being in the world or map coordinate frame, i.e., ${}^{W}\boldsymbol{T}_{L_1}, \cdots, {}^{W}\boldsymbol{T}_{L_4}$. Although, the same trajectory can also be described by the relative transformations of consecutive poses, i.e., ${}^{W}\boldsymbol{T}_{L_1}, {}^{L_1}\boldsymbol{T}_{L_2}, \cdots, {}^{L_3}\boldsymbol{T}_{L_4}$. Graph-based SLAM or loop closure-enabled algorithms may use graph optimization methods and prefer this kind of trajectory representation. The black arrows in Figure 2.1(b) illustrate the relative transformations to the world frame and between consecutive poses.

It should be noted that the map created and used by the SLAM algorithm is usually

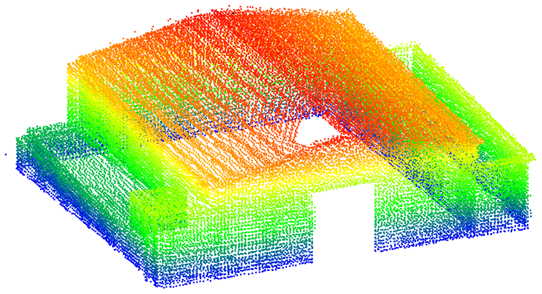


(a) Coordinate system following the right-hand rule

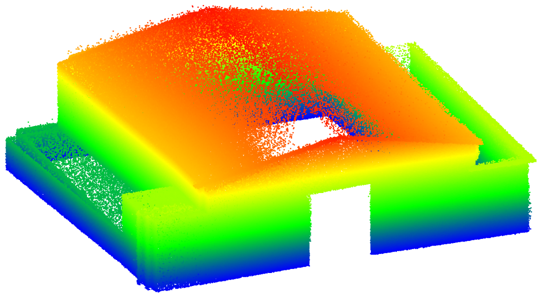

$\{L_3\}$
$^{L_2}\boldsymbol{T}_{L_3}$
$\{L_2\}$
$^{L_3}\boldsymbol{T}_{L_4}$
$^{L_1}\boldsymbol{T}_{L_2}$
$^{W}\boldsymbol{T}_{L_3}$
$^{W}\boldsymbol{T}_{L_2}$
$\{L_4\}$
$\{L_1\}$
$^{W}\boldsymbol{T}_{L_4}$
$^{W}\boldsymbol{T}_{L_1}$
$\{W\}$
Z
Y
X

(b) Four poses $\{L_1\}$ to $\{L_4\}$ are described as transformations to the world coordinate system $\{W\}$ and relative to each other.

Figure 2.1: The right-hand coordinate system is used in LiDAR-SLAM systems as illustrated in (a). A sequence of poses is described as relative transformations (b). The coordinate systems represent the poses and the black arrows illustrate the relative transformations between the poses.



(a) Sparse map of a building created by a SLAM algorithm and used for online localization



(b) Dense 3D reconstruction created offline from accumulated raw LiDAR scans using the computed poses from the SLAM algorithm in (a)

Figure 2.2: Comparison of a map from a SLAM algorithm (a) with the 3D reconstruction created with raw LiDAR scans and the SLAM poses (b). The ground was removed for illustrative purposes in both figures.

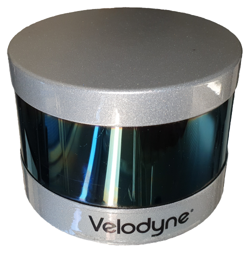
quite sparse and only contains points or point features necessary for accurate localization. Typically, SLAM algorithms use some kind of voxelization techniques and do not retain all information captured by the sensors. Otherwise, this would lead to rapidly growing memory consumption and higher computational requirements when estimating the pose in very large maps. Figure 2.2 shows a comparison between a SLAM map and a dense 3D reconstruction of the same building. The map (a) is voxelized and only one point per voxel (centroid) is kept during the map creation process, which is sufficient for pose estimation but leads to a sparse environment model. For the map creation, a voxel edge length of $40\,\mathrm{cm}$ was used. On the walls and roof, the regular distance between the voxel centroids can be seen. Denser regions are due to multiple occupied voxels behind each other, i.e., "double walls". A full 3D reconstruction can be created by transforming and accumulating all raw LiDAR scans $\mathcal{P}^{L_i}$ in the world/map frame (b) with the corresponding SLAM poses ${}^{W}\boldsymbol{T}_{L_i}$.

On the left corner of the point cloud in Figure 2.2(a), the "double wall" effect can be seen. Due to wrong scan matches during the pose estimation, new scans are inserted into the map with a wrong transformation. A common challenge in long-term localization and mapping is the maintenance of the map while preventing deterioration. The more scans are captured during exploration and inserted into the map, the worse the quality of the map becomes. This gets more obvious if the robot stays in the same environment during exploration. Points are continuously inserted into the map to correct prior mistakes during mapping, but also to add new parts to the map. The contributions of this thesis aim at reducing map deterioration by leveraging prior knowledge about the environment.

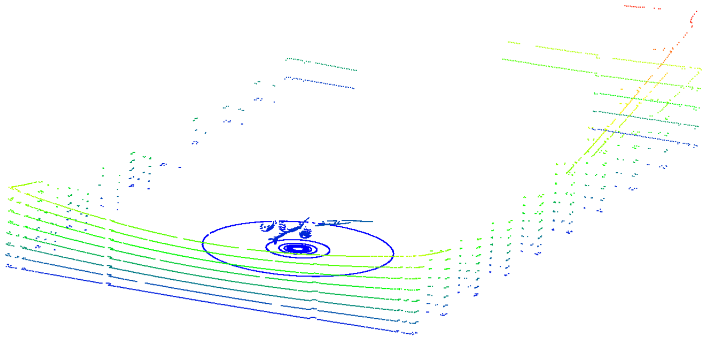
2.1.2 Point cloud acquisition

Point clouds can be generated with depth sensors, such as RGB-D cameras or LiDAR sensors. For RGB-D, three different types of depth perception currently exist: structured light (active), stereopsis (passive), and laser scanning (active). The first active RGB-D cameras projected structured light on surfaces to perceive depth. By measuring the deformation of a known light pattern in the scene, depth values can be computed. Structured light cameras usually have a low range of a few meters due to the limitations of the light projector. Other RGB-D cameras use stereopsis for depth perception. The disparity between two calibrated RGB cameras is used to estimate the depth for each pixel. Hence, it does not have an active depth sensor, but can generate higher resolution RGB-D images and can estimate depth values at a longer range compared to structured light sensors. Currently, there is a trend towards active RGB-D cameras with laser scanning technology. Depth is measured with a solid-state LiDAR sensor allowing for higher measurement ranges compared to structured light sensors. Due to the active depth sensor, range values can be measured with much higher accuracy compared to the disparity computation of cameras using stereopsis technology. A detailed comparison was conducted by Laurenço et al. [13]. They compared the three different types of depth perception using Intel RealSense RGB-D cameras.

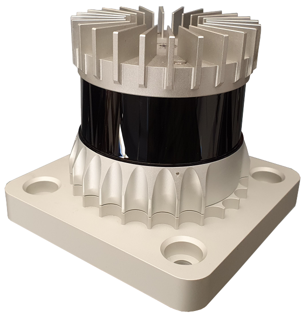
Two types can be differentiated concerning depth-only sensors: solid-state and spinning LiDAR. The major difference between a solid-state sensor and a spinning LiDAR is that the former produces a more dense depth output compared to spinning LiDAR sensors at the cost of a lower horizontal FoV. Spinning LiDARs usually cover the full 360° range and solid-state only a fraction of it. Different types of solid-state sensors exist, such as Micro-Electro-Mechanical Systems (MEMS), phased array, or flash LiDAR. MEMS steer a single laser beam with small mirrors, while phased array and flash LiDARs do not contain moving parts. Due to the smaller FoV, the emitted light is higher concentrated in one direction and therefore needs to be less energetic compared to spinning LiDAR sensors to comply with Class 1 eye safety. It implies that no eye damage must occur, even when putting the eye without blinking right in front of the sensor for several seconds. Spinning LiDAR sensors consist of a rapidly rotating mirror, refracting the laser beam horizontally and vertically in case of 360° 3D spinning LiDAR sensors. When putting the eye right in front of it, no eye damage occurs even with a higher energetic laser beam compared to solid-state due to the quick 360° rota-



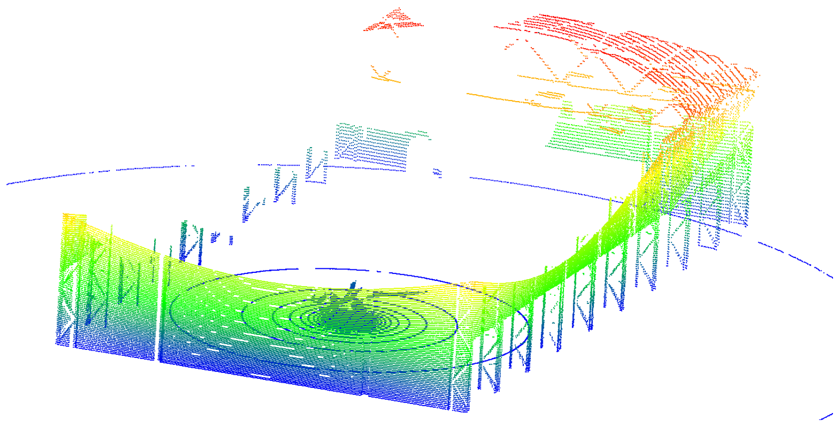
(a) Velodyne VLP-16L "Puck Lite"



(b) Single scan of a VLP-16 (a)



(c) Ouster OS1-128



(d) Single scan of an OS1-128 (c)

Figure 2.3: Illustration of two spinning 360° 3D LiDAR sensors from the manufacturers Velodyne (a) with 16 scan lines and Ouster (c) with 128 scan lines. (b) and (d) show simulated LiDAR scans of a hall. The virtual sensors follow the technical specifications of the manufacturers. Lower scan lines are displayed in blue and higher scan lines in red color.

tions. This allows for measuring higher distances with spinning LiDAR sensors compared to solid-state. Hence, eye safety is still one of the main hurdles for solid-state LiDAR to catch up.

Figure 2.3 shows examples of 360° 3D LiDAR sensors. A Velodyne VLP-16 (a) with 16 vertical scan lines and an Ouster OS1-128 (c) with 128 vertical scan lines. Exemplary virtual LiDAR scans of the two sensors can be seen in (b) and (d). The OS1-128 has a much higher vertical point density, which significantly increases the scene understanding ability. Also, the OS1-128 LiDAR has a higher vFoV, which can be seen on the higher wall in (d). The size of the sensors is very similar, despite the more recently released Ouster LiDAR sensor having 8 times the number of vertical scan lines. Table 2.1 shows a comparison of their technical specifications according to the official datasheets from the manufacturers. The OS1-128 has a slightly larger vFoV with 45° compared to 30° of the VLP-16. Both detect points up to a distance of 100 m and weight around 500 g, which make both suitable for mounting on larger

Table 2.1: Overview of the technical specifications of a Velodyne VLP-16L "Puck Lite" and Ouster OS1-128 LiDAR. *The results are presented for the single return mode.

LiDAR	#scan lines	vFoV	Range	Rotation Rate	Points per second*	Weight
Velodyne VLP-16L [14]	16	$\pm 15°$	up to $100\,\mathrm{m}$	$5\,\mathrm{Hz} - 20\,\mathrm{Hz}$	300 000	$590\,\mathrm{g}$
Ouster OS1-128 [15]	128	$\pm 22.5°$	$0.3 - 100\,\mathrm{m}$	$10\,\mathrm{Hz}$ or $20\,\mathrm{Hz}$	2 621 440	$447\,\mathrm{g}$

UAVs. The weight specified for the Ouster sensor is not considering the mounting bracket shown in Figure 2.3(c), which can be replaced with a lighter alternative. The main difference between the sensors is the number of points that can be measured per second. While the VLP-16 may be sufficient for SLAM applications, the OS1-128 has a much higher point density, which makes it more suitable for object detection in single LiDAR scans. However, processing so many points in real-time requires a powerful computational unit, which may not be carried by a UAV depending on the point cloud processing algorithms. Both sensors have the option to not only return one point per measurement but the strongest and the second strongest. This may be useful when measuring through transparent glass, fog, or dust. By default, the strongest point is saved for each measurement in single return mode. However, especially in foggy environments, the second strongest measurement may reveal the true underlying structure and therefore increase object detection capability, e.g., pedestrians and other traffic participants.

In this thesis, a VLP-16 "Puck Lite" LiDAR sensor is used for the majority of the experiments in simulation and also in a real environment. The Ouster OS1-128 is used in simulation and offline processing to give an impression of the performance of the proposed algorithms if a LiDAR sensor with more scan lines is used.

2.1.3 Point cloud and mesh representation

The environment can be represented as different digital models. One of the most common representations is the point cloud and a mesh. Both types are explained in more detail in the following sections.

2.1.3.1 Point cloud

A point cloud $\mathcal{P} = \{p_1, \cdots, p_n\}$ is a collection of n individual points p. The most common properties of a 3D point p are listed as follows:

- Position (required): x, y and z coordinate of the point p in 3D space
- Color (optional): Red, Green and Blue values of the point p
- Intensity (optional): Intensity value of the scan point, e.g., obtained from the LiDAR sensor
- Normal (optional): A three-dimensional normal vector defining a local plane

The position of the points is the only required information. In addition, each point may feature color information in RGB format. Typically, the values are specified in 8-bit, ranging from 0 to 255. The color information may be measured directly by the sensor, e.g., RGB-D, or can be assigned afterward to the points. For example, by mapping RGB image data to the point cloud, leveraging calibration data. An intensity value is directly measured for each point by a LiDAR sensor. It is a measure of reflectivity and depends on the angle the LiDAR beam hit the surface and surface properties. For example, some black materials may fully absorb the beam and no point will be measured. Other materials and surface colors have a strong reflectivity. A higher intensity value means that the LiDAR sensor picked up a strong return signal and vice versa. Each point may also have normal information. The normal vector consists of three 32-bit (typically) real values, describing a local plane at the point location. Normal vectors can be computed for each point by considering their local neighborhood. Rusu [16] described a method based on least-square plane fitting. It first computes the 3D centroid of the local neighborhood around the point, followed by the covariance computation. The eigenvector with the smallest eigenvalue is then considered as point normal. The biggest challenge here is to find the correct orientation of the normal. If a viewpoint or sensor origin is known, normal vectors are typically flipped towards it. If it is unknown, the point normals may be wrongly oriented and have a negative impact on further processing steps. Normal vectors are essential for some surface reconstruction algorithms, e.g., Poisson Surface Reconstruction [17], which computes a triangular mesh from an input point cloud with normals. Also, point feature extraction algorithms use normal vectors to compute point descriptors for point cloud registration [18].

Point clouds are common not only for environmental representations but also for raw LiDAR scans. By default, each LiDAR scan has the sensor center as the coordinate frame (origin). In order to properly reconstruct an environment consisting of accumulated point clouds, the transformation between the point clouds or to a global frame must be known, i.e., ${}^{W}\boldsymbol{T}_{L_1}$ for $\mathcal{P}^{L_1}$, ${}^{W}\boldsymbol{T}_{L_2}$ for $\mathcal{P}^{L_2}$, and so on. Now the point clouds can be transformed and accumulated in the common frame $\{W\}$. The transforms can be estimated with LiDAR-SLAM algorithms.

Point clouds can be stored in different file formats. One of the most common is the Point Cloud Data (PCD) or ASCII (ASC/PTS) format. While ASC is very useful for manually inspecting point cloud data, the binary encoding of a PCD file reduces the required file size, which is especially interesting for very large point clouds. As an example, a point cloud consisting of 100 million points requires 3.61 GB when saved in ASC format, while the binary PCD file of the same content only requires 1.60 GB (44.3%). This ratio scales linearly with the number of points.

CloudCompare[1] is an open-source point cloud processing software, which also features a Graphical User Interface (GUI). It allows for quick point cloud visualization, manipulation, cropping, registration, and also mesh surface reconstruction. For the integration of point cloud handling and processing in C++ or python code, the Point Cloud Library (PCL) [19] is the current state-of-the-art. It has a collection of seemingly all basic point cloud processing

[1] https://www.cloudcompare.org

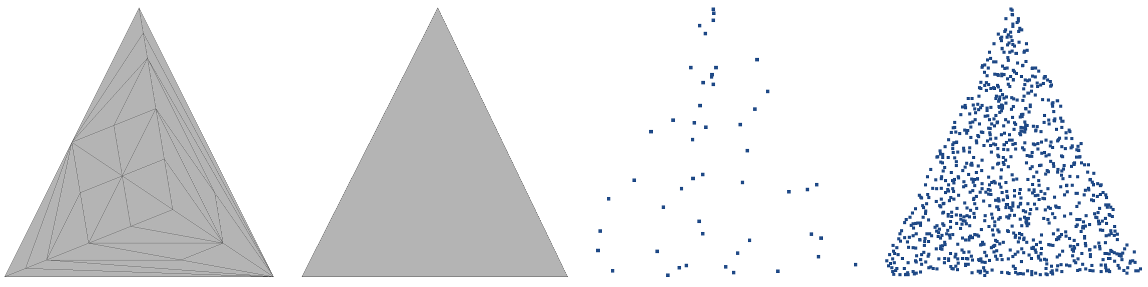


Figure 2.4: Description of the same triangular surface in different representations: (a) subdivision in multiple sub-triangles, (b) single triangle, (c) sparse uniform point sampling, (d) dense uniform point sampling

methods, such as smoothing, coordinate transformation, cropping, registration, normal estimation, and so on. It can be integrated as a library in C++ code to efficiently process large point clouds and LiDAR sensor data in real-time. The PCL library can be found as a dependency in many existing LiDAR-SLAM algorithms. Its popularity partially arises due to the 3-clause BSD license, which allows its usage for commercial and research applications. In this thesis, the functionality of the PCL library is a basic part of the presented contributions.

2.1.3.2 Triangular mesh

A triangular mesh is composed of vertices and faces. A vertex $v_i = [i, x, y, z]$ contains the index i, and the x, y and z position. Essentially, it is an enumerated 3D point. All vertices $\mathcal{V}$ in the mesh can then be described as a collection $\mathcal{V} = \{v_1, \cdots, v_n\}$, consisting of n vertices. In a 3D triangular mesh, a face is a 3D triangle defined by three vertices in $\mathcal{V}$. Hence, a face can be written as $f_j = [j, v_a, v_b, v_c]$. j is the index of the face and v_a, v_b, and v_c are the vertices with indices a, b and c, which define the 3D triangle. A collection of m faces is therefore $\mathcal{F} = \{f_1, \cdots, f_m\}$. The triangular mesh can then be expressed as $\mathcal{M} = [\mathcal{V}, \mathcal{F}]$. This is a very basic definition. Eventually, it can also include texture information for each vertex or face and normal information. Some mesh file formats even skip the index value i for the vertices. Instead, a, b and c in the definition of a face f refer to the corresponding line numbers in the file where the vertices are defined. This reduces the file size even further.

Triangular meshes can be saved in various file formats, e.g., Stanford Polygon (PLY), Stereolithography (STL), or Object (OBJ) among many other open and proprietary formats. As for point clouds, the vertex and face information can be saved in ASCII and binary format. MeshLab [20] is open-source software, featuring many methods for mesh visualization and processing. It supports many open mesh file formats and can also be used for conversion.

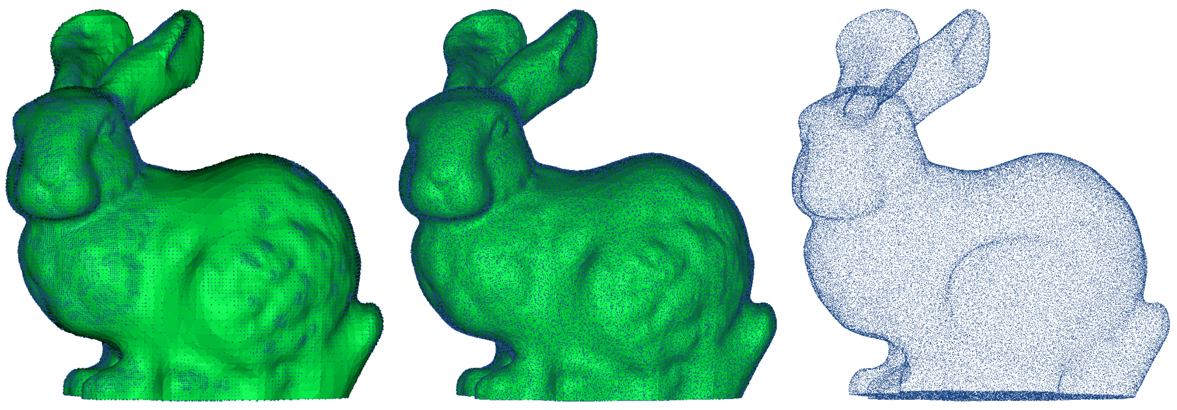
Figure 2.4 illustrates different representations of a triangular surface. (a) consists of 32 faces and (b) shows the same surface, described with only a single face. (a) can be reduced to the single face (b) without a loss of information, assuming that all faces of (a) and (b) lie on the same 3D hyperplane, i.e., without curvature. This mesh simplification can be performed with Quadratic Edge Collapse Decimation [21], for example. (c) and (d) are uniformly sampled point clouds on the triangle surface, consisting of 50 and 1000 points, respectively. 50 points are barely enough to estimate the triangular shape, while it is clearly visible with 1000 points. Comparing these representations, it can be concluded that a triangular mesh is a much more efficient and complete way to describe 3D surfaces. Hence, one can see a triangular mesh as an infinitely dense sampled surface, where the triangles are implicitly described by the three enclosing vertices. If an environment or object model exists in form of a CAD format or mesh file, the conversion to a point cloud inevitably leads to a loss of surface information and should be avoided. On the other hand, if a dense and accurate point cloud exists, the conversion to a mesh file may introduce errors. The main challenge here is to correctly combine neighboring three points forming a triangle/face. Poisson Surface Reconstruction [17] is an algorithm that takes a point cloud with pre-computed normals for all points as input and reconstructs the surface as a triangular mesh. It is, however, highly dependent on correctly oriented normals and an equal point cloud density.

Several programming libraries exist that support triangular mesh handling. The PCL library possesses basic mesh visualization capability. The libigl [22], Computational Geometry Algorithms Library (CGAL) [23] or Open3D [24] libraries feature extensive mesh generation, manipulation, and computational capabilities in C++ and partially Python. In this thesis, the libigl library is used for efficient mesh processing.

2.1.4 Point cloud registration

A LiDAR point cloud is usually recorded in the coordinate system of the sensor. When the position of the sensor is not tracked while moving it to the next recording position, the transformation between the two coordinate systems is unknown and the recorded point clouds can not be merged without additional computation and adjustment. Several applications exist for which a point cloud registration is required. Two or more point clouds were recorded in the same environment and have an overlap, meaning that some structures in both point clouds are identical. Typically, the larger this overlap is, the easier it is for the registration algorithm to align the two point clouds and the better is the result. A synonym for point cloud registration is therefore point cloud alignment. The necessity to merge several point clouds may arise from the goal of an improved 3D reconstruction of an object or a whole environment. For example, if the complete reconstruction of an object is desired, the LiDAR sensor needs to be positioned multiple times around the object to record point clouds. These individual point clouds then need to be merged to retrieve a single point cloud of the full reconstruction. Another application for point cloud registration is not targeting the reconstruction itself but rather aims at refining the pose, where the LiDAR sensor was positioned during the recording. A good reconstruction result and the proper knowledge of the LiDAR pose are tightly coupled since only a well-refined pose will lead to a good reconstruction. An example is the registration of a single LiDAR scan to a map. After registering a scan, the refined pose is the best estimate of the LiDAR sensor's position within the map.

Different types of point set registrations exist, e.g., *rigid* or *non-rigid* registration. In this thesis, only rigid registrations are considered where the transformation is a homogeneous 4×4 matrix with only a rotational and translational component without scaling. Non-rigid



(a) Stanford Bunny as triangular mesh with 112k faces and 56k vertices

(b) Overlay of the Stanford Bunny mesh with 100k uniform point samples

(c) Pure point-sampled Stanford Bunny

Figure 2.5: Illustration of the triangular mesh to point cloud conversion process. The Stanford Bunny as triangular mesh (a) is converted to a pure point-sampled representation (c).

registration would require an affine transformation with scaling and possibly shearing. Typically, scaling of point clouds from a LiDAR sensor is not required, since all points are in the same unit system after acquisition (i.e., metric or imperial). However, scaling is sometimes used in trajectory alignment methods for evaluation, e.g., Umeyama's method.

Point cloud registration is mostly performed between two point clouds, e.g., for LiDAR-SLAM applications or 3D reconstructions. However, some circumstances require the registration of a point cloud to a triangular mesh. For example, when the goal is the LiDAR-based relative localization to an object, whose model is directly available as a triangular mesh (e.g., as a CAD model). The mesh can be converted to a point cloud by uniform point-sampling of the faces, but this would effectively remove the surface information of the mesh model. Figure 2.5 illustrates the conversion process of the Stanford Bunny from a triangular mesh to a point cloud. The model initially consists of 112k faces and 56k vertices (a). Here, the efficient surface description becomes visible. In areas with higher curvature, more faces/vertices are required (front of the bunny). In contrast, flatter regions can be described with larger triangles, resulting in fewer faces and vertices (center). The surface is then uniformly sampled to generate a total of 100k points (b). Instead, the vertices of the mesh from (a) could be used directly as point cloud representation. However, the surface of larger triangles is then not well described in the point cloud model and this may result in poorer point cloud registration performance. The figure in (c) shows the final point cloud of the Stanford Bunny without normals, which may be used for point cloud to point cloud registration.

In the following, algorithms for feature extraction, and point cloud to point cloud/mesh registration without conversion are explained in detail.

2.1.4.1 Point cloud features

Feature extraction is an important process for many applications on point cloud processing, such as registration for 3D reconstruction or pose estimation. Feature extraction algorithms are dependent on the point cloud density/resolution and noise level. To improve reliable feature extraction, smoothing or outlier filtering methods may be used in a preprocessing step. However, in most SLAM applications, voxelization rather than smoothing operations are used to overcome the problem of growing map sizes and memory consumption in the course of the robot exploration.

The extraction of features for *global* registration mainly takes place when a kidnapped robot event occurred. In this case, no assumption about the position of the robot within the map is necessary. It may also be used to register the point cloud model of an object in a map, e.g., to find the position of the object or to count the present instances for inventory purposes. A traditional and commonly used point feature is the Fast Point Feature Histogram (FPFH) [25]. It creates descriptors from weighted, histogram-binned angular values of the connections between a query point and its neighboring points. It is a real-time capable version of Point Feature Histograms (PFH) [18]. Both methods require surface normals for each query point to compute the angular values. Recently, there is a shift from hand-crafted features, following deterministic and reproducible results to deep features, which may have some kind of randomness involved depending on the architecture. The randomness may be introduced already in the training process of the network by random weight initialization, or random shuffling of the training data. During inference, randomness is introduced by regularization techniques such as dropouts. One example of a neural network-based feature is the Fully Convolutional Geometric Feature (FCGF) [26].

The mentioned features may also be used for *feature-based local* point cloud registration. Most of these feature extraction algorithms are applied to dense representations of either a complete scene or individual objects, which are registered to the scene. In the context of LiDAR-SLAM however, the main challenge is to register a sparse incoming LiDAR scan to either a previous scan or to the map. To overcome the sparseness of single scans, Droeschel et al. [27] accumulate 2D LiDAR scans in a Micro Aerial Vehicle (MAV) frame. The motion of the MAV during acquisition is estimated with visual odometry of two stereo cameras. The resulting local maps then have a higher density than single scans and can be better integrated into the global map with an approach based on Iterative Closest Point (ICP) using all points to find correspondences.

Instead of using all points of the sparse scan for the registration to the map, feature detection algorithms have been developed, which leverage knowledge about the LiDAR sensor specifications, e.g., the number of vertical scan lines or horizontal angular resolution.

The LOAM framework, proposed by Zhang et al. [12], extracts corner and surface features directly from sparse 3D LiDAR scans. The extracted features are suitable for scan-to-scan (odometry) and scan-to-map matching (mapping). The correspondence estimation steps of the LOAM framework perform searches across several scan lines. Hence, for each scan point, it needs to be reconstructed from which scan line of the LiDAR sensor it originated. A requirement for this computation is knowledge of the technical specifications of the sensor, i.e., vFoV and the number of vertical scan lines. For a 3D point $p = [x, y, z]$, the elevation angle ϕ can then be determined with

$$\phi = \arctan\left(\frac{z}{\sqrt{x^2 + y^2}}\right) \cdot \frac{180}{\pi} \,. \tag{2.3}$$

With the elevation angle ϕ for the point p, it is now possible to reconstruct from which scan line of the LiDAR sensor p originated. The following equation assumes a zero-centered uniform beam configuration (i.e., equally distributed scan lines over the vFoV). For example, the scan lines need to be distributed from $-X$ to $+X$ degrees vFoV. It is not suitable for sensors, which have a varying scan line density (e.g., Ouster "Gradient") or when the scan lines are not centered around 0 degrees (e.g., Ouster "Below horizon").

$$l = \left\lfloor \left(\phi + \frac{v}{2}\right) \cdot \frac{n}{v} \right\rfloor \tag{2.4}$$

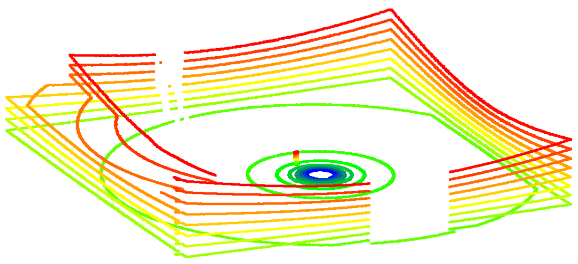
v is the vFoV in degrees and n is the number of scan lines of the sensor. l is the zero-based scan line number from which the point p originated. This equation can be adapted to be compatible with other types of sensors as well, e.g., varying scan line density or non-centered. Figure 2.6(a) illustrates the result of the scan line sorting. The simulated scan shows an empty indoor environment. The LiDAR sensor was placed on the ground during acquisition. Blue color correspond to lower and red color to higher scan lines. The typical ring shapes of a $360°$ LiDAR sensor in the lower scan lines on the ground are visible.

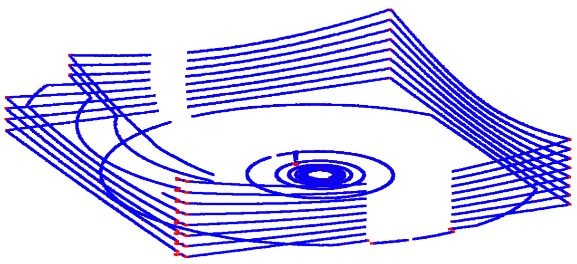
The feature extraction process of the LOAM framework [12] computes corner and surface features based on the curvature of each point. With the points sorted into the corresponding scan lines, it is now possible to compute the curvature for each point by taking the position of preceding and succeeding points in the same scan line into account. The curvature c_p for a point p is computed with the following equation:

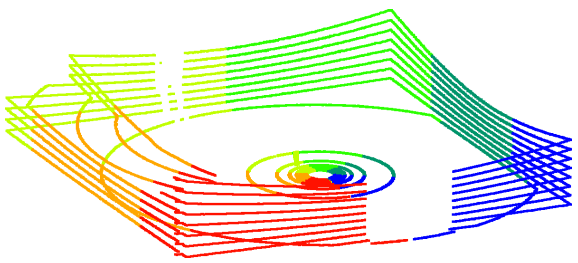
$$c_p = \sum_{a=[x,y,z]} \left(\sum_{i=-5}^{-1} p_i(a) - 10 \cdot p(a) + \sum_{i=1}^{5} p_i(a) \right)^2 \tag{2.5}$$

p is the current point, and p_i represents the five preceding and succeeding points. The sum of squares for each axis $[x, y, z]$ is then the scalar curvature value c_p for the point p. Figure 2.6(b) shows the scan points colored according to their curvature value. Low curvature is represented by blue color and high curvature is represented by red color. It can be seen that mostly the corners of the room have the highest curvature along the scan lines.

After the curvature computation, the corner and surface features are extracted. Assuming $360°$, the scan is first divided into six horizontal sections as illustrated in Figure 2.6(c). In each section and for each scan line, the two sharpest ($\mathcal{P}_\mathcal{E}$), 20 less sharp corner features ($\mathcal{P}_{\mathcal{E}_\text{less}}$), and the four flattest surface features ($\mathcal{P}_\mathcal{H}$) are extracted. A curvature threshold of 0.1 is used to separate corner ($c > 0.1$) and surface ($c < 0.1$) features. The sharpest and flattest features are determined after sorting according to the curvature values (see Fig. 2.7). Less sharp features


(a) Scan points colored according to their scan line (blue = lower scan line, red = higher scan line)


(b) Scan points colored according to their curvature value computed with Equation 2.5 (blue = low curvature, red = high curvature)


(c) Division of the scan into six sections

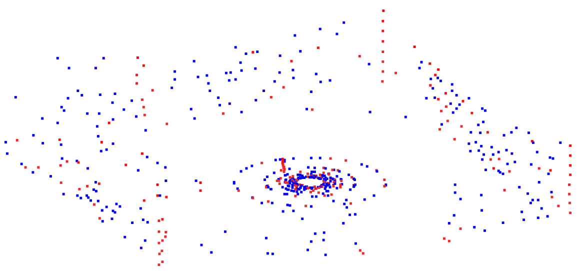

(d) Extracted sharp corner and flat surface features

Figure 2.6: Visualization of the feature extraction process from scan line sorting (a), curvature computation (b), section division (c) to the extracted features per section and scan line (d). The scan was generated in an indoor environment with a virtual VLP-16 LiDAR sensor from a simulation. Gaussian noise was added to simulate ranging errors.

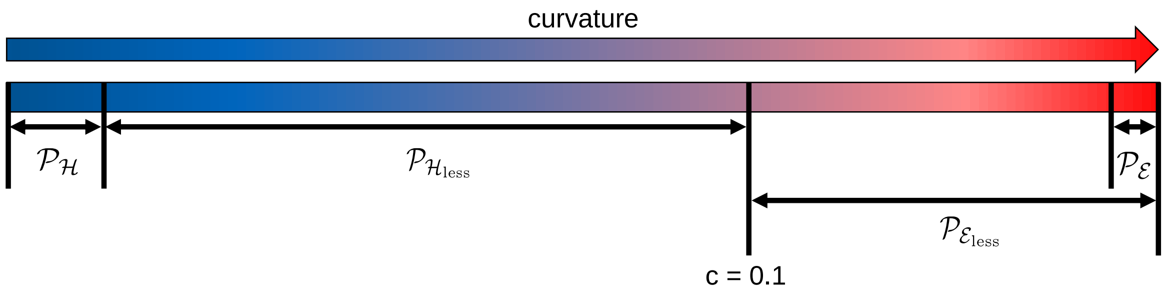

curvature
$\mathcal{P}_{\mathcal{H}}$
$\mathcal{P}_{\mathcal{H}_{\text{less}}}$
$\mathcal{P}_{\mathcal{E}}$
$\mathcal{P}_{\mathcal{E}_{\text{less}}}$
c = 0.1

Figure 2.7: Division of the scan points of one section and one scan line into the four types of features. The scan points of the section are sorted ascendingly according to their curvature value.

can be understood as the 20 sharpest points, which also include the two sharpest features. All remaining points are assigned to the category of less flat surface features ($\mathcal{P}_{\mathcal{H}_{\text{less}}}$), which can be seen as something in between flat and less sharp features. The division into sections ensures that features are extracted from all parts of the 360° scan. Otherwise, there might be the chance that features are clustering around one object in the environment, which is unsuitable for the SLAM algorithm. Figure 2.6(d) shows the resulting sharp corner (red) and flat surface features (blue). The corners of the room are clearly well represented by equally-spaced corner features. Surface features are also equally distributed on the walls and ground. However, the forced extraction of sharp corner features in each section of the scan and for each scan line results in incorrect/too many corner features. This effect can be seen on the ground and on the walls, where practically no corners are to be expected. Nevertheless, this method guarantees a stable number of extracted features in each category and has proven sufficient for most SLAM scenarios.

The presented corner and surface feature extraction algorithm is used for the experiments in this thesis due to its reliability and real-time performance.

2.1.4.2 Point cloud to point cloud registration

Point cloud to point cloud registration aims at finding a transformation $\boldsymbol{T}$, which best aligns a source point cloud $\mathcal{P}_s = \{p_{s_1}, p_{s_2}, \cdots, p_{s_l}\}$ to a target point cloud $\mathcal{P}_t = \{p_{t_1}, p_{t_2}, \cdots, p_{t_n}\}$. A typical sequence of steps is as follows:

1. Feature detection
2. Feature description
3. Correspondence estimation (feature matching)
4. Correspondence rejection
5. Optimization

Steps 1 and 2 are sometimes also called *feature extraction*.

Feature detection is the process of selecting a subset of keypoints from $\mathcal{P}_s$ and $\mathcal{P}_t$. These are points that best describe the scanned part of the environment, e.g., corners, edges, or surfaces as described in Section 2.1.4.1. Detected features should also have a high chance for redetection in other point clouds if the environment is the same. This property of a feature detection algorithm is also called *repeatability*.

During **feature description**, the local neighborhood of points is considered for each keypoint to compute a X dimensional feature descriptor vector. It consists of up to several thousand integer values ($\mathbb{Z}^X$) or real ($\mathbb{R}^X$) numbers. A common method is to estimate surface normals from neighboring points and then convert the normals to feature vectors.

During **correspondence estimation**, matching pairs of features are detected between $\mathcal{P}_s$ and $\mathcal{P}_t$ based on a nearest neighbor search in feature space. Alternatively, a nearest neighbor search can be performed directly on the points in $\mathcal{P}_s$ and $\mathcal{P}_t$, skipping the feature detection and/or description process of steps 1+2 to find correspondences.

Correspondence estimation is known to suffer from outliers, which are incorrect point or feature matches forming a correspondence. These can be filtered with **correspondence rejection** methods. The reason for removing incorrect correspondences is to improve the input data quality for the following optimization in step 5. A variety of correspondence rejection methods exist. Some apply a maximum or median distance threshold to each correspondence in either point or feature space. Other methods eliminate correspondences with duplicate point/feature usages or by performing a nearest neighbor consistency check from $\mathcal{P}_s$ to $\mathcal{P}_t$ as well as from $\mathcal{P}_t$ to $\mathcal{P}_s$. Only if the correspondence pair is confirmed in both directions, it is retained as a valid match. More advanced rejection algorithms estimate a RANSAC model to reject outliers, use thresholds on angles of corresponding surface normals, or only keep a number of correspondences based on an overlap parameter between the two point sets [28].

Table 2.2: Overview of the required steps for global and local registration of two 3D point clouds

Registration type	Steps for point cloud registration				
	1. Feature detection	2. Feature description	3. Correspondence estimation	4. Correspondence rejection	5. Optimization
Global and feature-based local	x	x	x (in $\mathbb{Z}^X$ / $\mathbb{R}^X$)	x (in $\mathbb{Z}^X$ / $\mathbb{R}^X$)	x (in $\mathbb{Z}^X$ / $\mathbb{R}^X$)
Local	optional	-	x (in $\mathbb{R}^3$)	x (in $\mathbb{R}^3$)	x (in $\mathbb{R}^3$)

Every step from 1 to 4 is critical since the optimization of step 5 is highly dependent on the correct correspondences. Mistakes in earlier steps will accumulate through later stages, i.e., inliers in step 4 may be wrongly removed and outliers may be kept. The **optimization** in step 5 aims at minimizing the residuals of an objective (cost) function. It may be as simple as minimizing the Euclidean distances between the correspondences by refining the rigid transform $\boldsymbol{T}$. The transform aligns the source point set $\mathcal{P}_s$ to the target point set $\mathcal{P}_t$. In the case of point features, e.g., corner or surface points, more advanced cost functions minimize line or planar distances, respectively.

Depending on the accuracy of the initial guess transformation $\boldsymbol{T}$ between $\mathcal{P}_s$ and $\mathcal{P}_t$, one speaks of *global* or *local* registration. Table 2.2 shows a comparison of the required steps for global and local registration of two 3D point clouds. Global registration does not require a good initial guess, but instead fully relies on the proper feature extraction and correspondence estimation (steps 1-4) for the optimization in step 5. The key to this is the computation of feature descriptors in step 2 and the correspondence estimation in feature space (step 3). Some feature-based local registration methods combine the 3D point coordinates with feature information, resulting in a descriptor vector, e.g., ICPIF [29]. It then follows the same steps as for global registration. In contrast, conventional local registration does not require feature description and can therefore perform correspondence estimation, rejection, and optimization in point space $\mathbb{R}^3$. Neighboring points of the point sets $\mathcal{P}_s$ and $\mathcal{P}_t$ can be directly used to form correspondences.

With a good initial guess and a rough point cloud to point cloud alignment, the transformation can be refined with a local registration method. The initial guess may either be known or can be acquired using a global registration algorithm. The most popular method for local point cloud to point cloud registration is ICP. It does not require a prior feature extraction but directly operates on the points. The first step is the correspondence estimation between two point sets. An exhaustive search would have the complexity $O(n)$ when trying to find the exact nearest neighbor of a single query point in a target point cloud. To quickly find the point correspondences between a source and target point cloud, *k*-d trees are frequently used. A *k*-d tree is a space-partitioning structure with complexity $O(\log n)$ to speed up nearest neighbor or radius searches in *k*-dimensional space. It is built by finding the median along one data dimension, splitting this space in half, and recursively repeating this process until a certain depth level is reached or the leaf nodes contain a certain number of points. Often it is sufficient to find the approximate nearest neighbors rather than performing an exhaustive search, which additionally speeds up the computation. This can be achieved by not traversing down the *k*-d tree to the last leaf node.

The first formulation of ICP was suggested by Chen et al. [30]. It essentially computes the nearest neighbors between two point clouds and iteratively minimizes their distance by repeating correspondence estimation and optimization steps until a convergence criterion is met, e.g., the Mean Squared Error (MSE) is below a threshold. The objective function [30], [31] to be minimized is denoted as

$$\boldsymbol{T} \leftarrow \arg\min_{\boldsymbol{T}} \sum_{i} w_i ||\boldsymbol{T} \cdot p_{s_i} - p_{t_i}||^2 \,. \tag{2.6}$$

Between a target point set $\mathcal{P}_t = \{p_{t_1}, p_{t_2}, \cdots, p_{t_n}\}$ and a source point set $\mathcal{P}_s = \{p_{s_1}, p_{s_2}, \cdots, p_{s_l}\}$, a point p_{s_i} is transformed with a matrix $\boldsymbol{T}$ to find the closest point p_{t_i} in $\mathcal{P}_t$. $\boldsymbol{T}$ is then iteratively optimized to minimize the correspondence distances between $\mathcal{P}_t$ and $\mathcal{P}_s$. w_i is a binary operator, which is set to 0 when the correspondence distance is larger than an outlier threshold and set to 1 if it is below. This formulation is known as the point-to-point variant since it only uses point information in its optimization.

Chen et al. [30] also proposed a point-to-plane variant of ICP, improving local registration performance. It leverages additional normal information for each point. The optimization formulation is adjusted as follows:

$$\boldsymbol{T} \leftarrow \arg\min_{\boldsymbol{T}} \sum_{i} w_i ||\eta_i \cdot (\boldsymbol{T} \cdot p_{s_i} - p_{t_i})||^2 \tag{2.7}$$

The formulation is very similar, but a scaling factor η_i is introduced, which scales the residual vector by the surface normal of p_{t_i}. Evaluations have shown an improvement compared to the point-to-point variant. However, it requires the existence or computation of surface normals for each point in the target point cloud $\mathcal{P}_t$.

Rusinkiewicz [32] proposed the use of a symmetric objective function, which considers the normals of the target as well as source point cloud at a similar computational complexity and a wider convergence basin. Segal et al. [31] then extended this approach by incorporating the plane-to-plane distance. The authors named the method Generalized-ICP (G-ICP). It adds a probabilistic model to the optimization step shown in Eq. 2.6, but also requires normals for the points. In the original publication, G-ICP outperforms standard point-to-point and point-to-plane ICP on real and simulated LiDAR data. Other ICP variants, e.g., Trimmed-ICP (TrICP) [28], Cluster-ICP (CICP) [33], and Go-ICP [34] performing optimization in a global search space have been proposed. Furthermore, the ICPIF algorithm [29] uses invariant features, e.g., shape descriptors for the correspondences.

Also, learning-based methods have been proposed recently, namely Deep Closest Point (DCP) [35] using DCGNN features [36] and Fast Global Registration (FGR) [37]. The latter claims to perform global registration with a shorter execution time than ICP. PointNetLK [38] employs the Lucas & Kanade algorithm for point cloud registration by leveraging PointNet [39] as a learnable function. Teaser++ [40] employs a truncated least-squares cost function, making the registration more robust against outlier correspondences. Yin et al. proposed LocNet [41]. The method leverages siamese networks for global coarse registration with ICP refinement.

In this thesis, the initial guess is assumed to be sufficient, and therefore only local registration is applied to point clouds. Due to the absence of normals and sparseness of the raw LiDAR data, the standard point-to-point ICP algorithm is applied.

2.1.4.3 Point cloud to triangular mesh registration

In contrast to point cloud to point cloud registration, some additional steps have to be taken for point cloud to triangular mesh alignment. As previously discussed, an existing triangular mesh may be converted to a point cloud and the registration methods of Section 2.1.4.2 may then be applied. However, this would remove surface information, which can be exploited especially in larger meshes with flat surfaces (e.g., walls, ceiling, and ground). These can be efficiently described with few faces.

Similar to point cloud to point cloud registration, the goal is to minimize the distance between the point cloud and the triangular mesh surface. Also here, the distance between each query point and the mesh can be seen as a straight line in 3D space. Since the mesh surface is implicitly defined as triangular faces by the enclosing vertices, the computation of virtual points on the faces is required as the nearest counterpart for each point in the query point cloud. Let $\mathcal{P}_v = \{p_{v_1}, p_{v_2}, \cdots, p_{v_n}\}$ be the virtual point set with n points on the mesh surface consisting of m faces/triangles and let $\mathcal{P}_s = \{p_{s_1}, p_{s_2}, \cdots, p_{s_n}\}$ be the query (= source) point cloud. The search for the nearest virtual point p_{v_i} on the mesh surface for each query point p_{s_i} can then be divided into two phases:

- **Phase 1**: Find the closest triangle on the mesh to the query point p_{s_i}
- **Phase 2**: Find the virtual point p_{v_i} on the triangle from step 1 closest to p_{s_i}

The number of points in $\mathcal{P}_v$ is implicitly determined by the size of $\mathcal{P}_s$, since a virtual point p_{v_i} needs to be found for each query point p_{s_i}.

Phase 1 The first step is to find the closest triangle/face on the mesh for each point p_{s_i} in the query point cloud. The naïve approach is to compute the closest virtual point on all faces of the mesh for each query point and then the virtual point with the shortest distance is taken as the closest point on the mesh surface. However, this brute-force approach scales with complexity $O(m * n)$ for a number of faces m in the mesh for n query points. Assuming n to be constant, the brute-force approach has a linear complexity $O(m)$. The computational time required for large meshes with hundreds of thousands or even millions of faces is not tractable, nor is real-time capable for a larger number of query points.

In the area of computer graphics, several applications exist in which it is required to detect object collision in real-time. Often, it is sufficient to just detect *if* a collision between

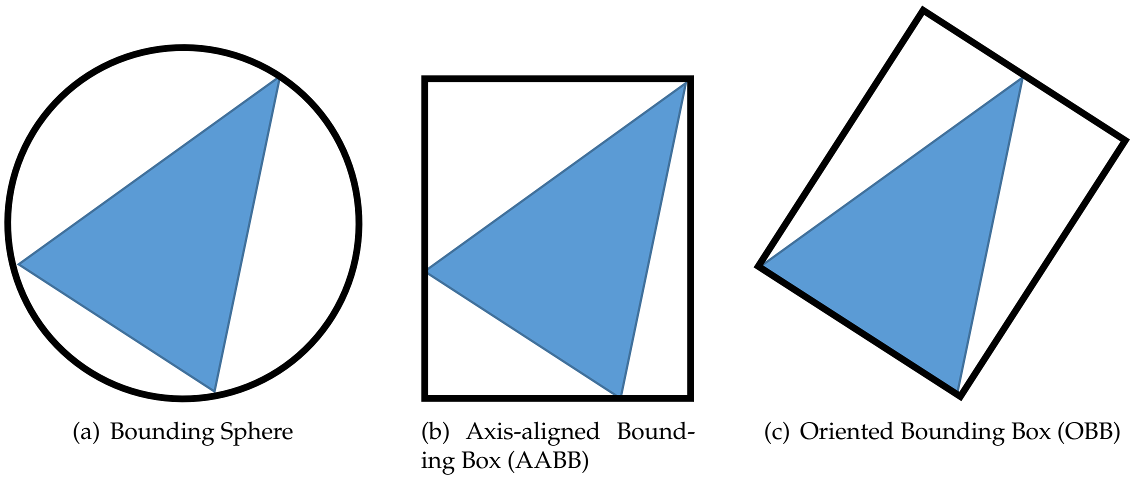
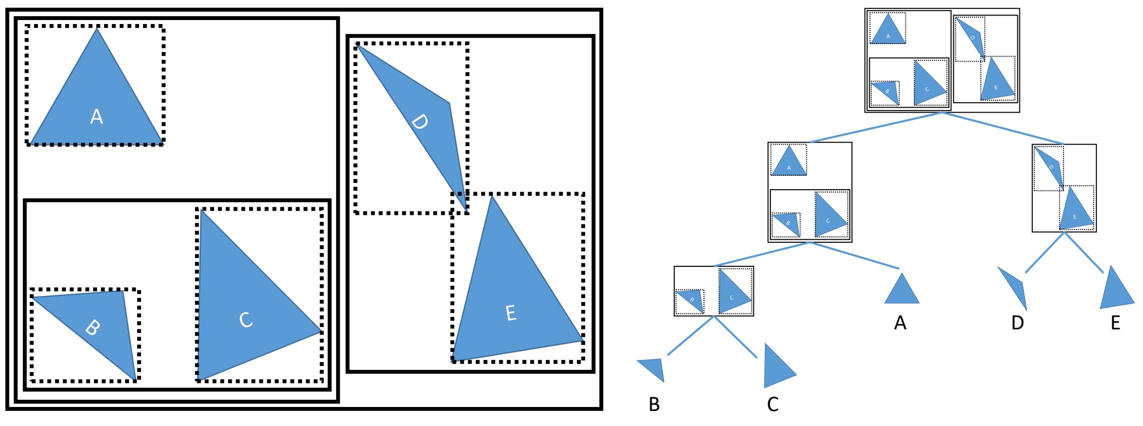

(a) Bounding Sphere
(b) Axis-aligned Bounding Box (AABB)
(c) Oriented Bounding Box (OBB)

Figure 2.8: Illustration of the most common bounding volumes used in collision detection applications. The triangles inside the bounding volumes have equal sizes. From left to right: better bound (less white space), more memory consumption, more expensive intersection/distance queries.

objects is taking place in a simulation or computer game, e.g., to trigger an explosion effect. Although, sometimes it may be required to determine the colliding sections of the objects, e.g., to visualize an impact or damage on the respective parts. With a large number of meshes and faces, a real-time capable collision detection using a brute-force approach is not realistic. For this reason, space partitioning techniques are used, i.e., Bounding Volume Hierarchies (BVHs) consisting of a hierarchy (tree) of bounding volumes. BVHs are comparable to *k*-d trees for point clouds.

The most common bounding volumes for collision detection are Bounding Spheres, Axis-aligned Bounding Boxes (AABBs), or Oriented Bounding Boxes (OBBs) as illustrated in Figure 2.8. While the triangle is more tightly bound from (a) through (c), intersection tests and distance queries are more complex. Also, the information to be stored about the bounding volume itself is higher. For example, for the Bounding Sphere (a) only the $[x, y, z]$ position of the sphere center and its radius need to be stored in 3D space, corresponding to four real values. The AABB (b) requires the minimum and maximum points of the box to be stored, corresponding to six values. OBBs (c) are defined by their center point (three values), orientation basis vectors ($3 \times 3 = 9$) and three scalars defining the dimensions of the box [42]. Each OBB, therefore, requires 15 real values to be stored, almost three times as much as for one AABB. A distance query between a point and an AABB can be efficiently performed by just computing the excess distance of a point outside the box to its minimum and maximum points, leveraging the axis alignment property. Bounding volumes may contain different kinds of basic primitives, i.e., polygons. Although, due to its convex properties, collision computation with triangles is computationally more efficient compared to the same number of polygons of higher order [43].

Individual bounding volumes can be organized in a hierarchy, i.e., one AABB may also contain several AABBs recursively. Like any hierarchical structure, a BVH has a tree degree (width), also called the branching factor, and a depth. A larger degree results in more chil-



(a) (b)

Figure 2.9: Hierarchical BVH tree using AABBs as bounding volumes. The tree contains five triangles as basic primitives. Reproduced and adapted from [44].

dren nodes but a more shallow tree, decreasing top-down traversal time while increasing required collision checks at each level. The opposite holds for low-degree BVHs, respectively. The most commonly used is a binary-degree BVH, due to its easy construction and traversal [44]. Van den Bergen [45] was one of the first to describe an efficient way to update and query binary AABB trees. For more information on how to construct or query BVH structures efficiently, the reader is referred to the book of Ericson [46].

Among all types of BVH, AABB trees are one of the most used structures for real-time collision avoidance due to their fast querying and construction. A major disadvantage, however, is that they can not be easily updated when the primitives inside were subject to a rotation, e.g., they belong to a moving object. In contrast, due to its orientation information, the bounding volumes inside an OBB tree can be updated with the new rotation without the need for recreating the whole structure.

Figure 2.9 illustrates an AABB tree with binary degree containing five triangles. A closest point computation will traverse down the tree by comparing the distance to the two AABBs of the current level. The next branches are chosen recursively by selecting the AABB with the shortest distance. Once the leaf AABB is found, the exact distance between the query point and the triangle can be computed (Phase 2).

In this thesis, AABB trees are used for fast point cloud to triangular mesh queries. The mesh object is static so there is no need for updating or recreating the AABB tree once it is constructed. Another term the reader might come across for BVH construction is *initialization*.

Phase 2 After querying the AABB tree to find the closest triangle with vertices ABC to a query point p_{s_i}, the closest virtual point p_{v_i} on the triangle to the query point needs to be computed. Despite it being an actual point on the triangle once found, it is termed *virtual* here, since it is no pre-existing point as in a real point cloud. To find the virtual point on the triangle closest to the query point, basic 3D geometry can be used. Ericson [47] describes several ways to efficiently determine the closest distance and the virtual point. If the orthogonal projection of p_{s_i} on the plane of ABC lies within the triangle, the closest point is the projection itself. Otherwise, it must lie on one of the edges of the triangle or maybe a vertex.

In the following, the notation of 3D geometry is adopted, which may coincide with the general notation used in this thesis for point cloud processing.

Let Q' be the projection of a point Q (e.g., the query point p_{s_i}) onto the plane of a triangle ABC. The barycentric coordinates $[u, v, w]$ of Q' can be determined directly from Q without computing the projection itself.

$$\begin{aligned} p_{QAB} &= \boldsymbol{n} \cdot (\boldsymbol{QA} \times \boldsymbol{QB}) \\ p_{QBC} &= \boldsymbol{n} \cdot (\boldsymbol{QB} \times \boldsymbol{QC}) \\ p_{QCA} &= \boldsymbol{n} \cdot (\boldsymbol{QC} \times \boldsymbol{QA}) \end{aligned} \tag{2.8}$$

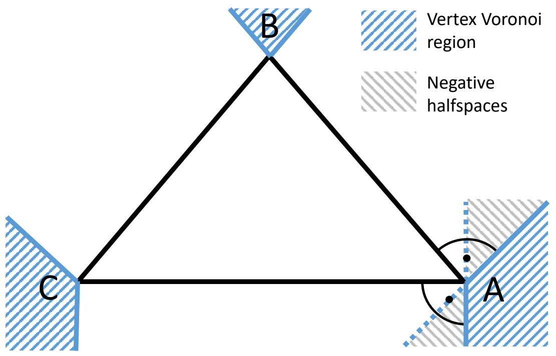
$\boldsymbol{n}$ is the unit normal vector on ABC computed with $\boldsymbol{n} = \dfrac{\boldsymbol{AB} \times \boldsymbol{AC}}{||\boldsymbol{AB} \times \boldsymbol{AC}||_2}$. p_{QAB}, p_{QBC} and p_{QCA} are then the proportions of the signed areas of the triangles QAB, QBC and QCA with respect to the signed area of triangle ABC. The barycentric coordinates can then be derived with:

$$\begin{aligned} p_{ABC} &= p_{QAB} + p_{QBC} + p_{QCA} \\ u &= \frac{p_{QBC}}{p_{ABC}} \\ v &= \frac{p_{QCA}}{p_{ABC}} \\ w &= \frac{p_{QAB}}{p_{ABC}} \end{aligned} \tag{2.9}$$

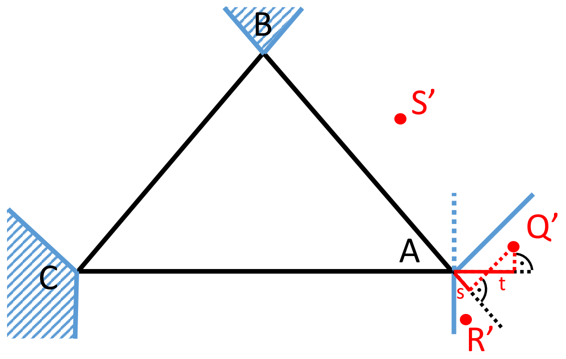
If all barycentric coordinates $[u, v, w]$ are positive, the projection of Q must lie within ABC and is, therefore, the closest point directly. Otherwise, it needs to be determined whether the closest point is on one of the edges or is a vertex. Instead of checking all edges and vertices for the closest point, it can be investigated in which Voronoi feature region the projected point Q' lies. The Voronoi regions of the vertices are determined by the intersection of the negative halfspaces of the planes through the vertices, which are in turn determined by the normal vectors of the adjacent triangle edges. Figure 2.10(a) illustrates the vertex Voronoi regions and the negative halfspaces at the vertex A. The intersections of the negative halfspaces are marked in blue color (vertex Voronoi regions). They mark the regions in which a projected query point on the plane of ABC has one of the vertices A, B, or C as the closest counterpart on the triangle. The white space outside the triangle and between the vertex Voronoi regions, mark the areas in which projected points have a virtual point on the edges $\boldsymbol{AB}$, $\boldsymbol{BC}$ or $\boldsymbol{CA}$ as the closest point on the triangle.

When evaluating the barycentric coordinates of three projected points Q', R', and S' in Figure 2.10(b), one will notice that one coordinate is always negative. However, only from the barycentric coordinates, it can not be easily determined whether the closest point is one of the vertices A/B or whether it lies on the edge $\boldsymbol{AB}$. Also, R' has a negative value in a different barycentric coordinate than Q', while still having A as its closest point.

Without having to compute the barycentric coordinates of the projected point Q' on the plane of ABC, it can be determined whether Q' lies within the vertex Voronoi region of A by



(a) Vertex Voronoi regions of a triangle ABC and the negative halfspaces at A



(b) Orthogonal projection of three points onto the plane of ABC

Figure 2.10: Illustration of the vertex Voronoi regions of a triangle ABC and exemplary orthogonal projections Q', R' and S'. Partially reproduced from [47].

determining s and t as illustrated in Figure 2.10(b):

$$\begin{aligned} s &= \frac{\boldsymbol{AQ} \cdot \boldsymbol{AB}}{\boldsymbol{AQ} \cdot \boldsymbol{AB} + \boldsymbol{BQ} \cdot \boldsymbol{BA}} \\ t &= \frac{\boldsymbol{AQ} \cdot \boldsymbol{AC}}{\boldsymbol{AQ} \cdot \boldsymbol{AC} + \boldsymbol{CQ} \cdot \boldsymbol{CA}} \end{aligned} \tag{2.10}$$

If s and t are negative, Q' definitely lies in the vertex Voronoi region of A and the closest point to Q/Q' is the vertex A. In fact, s and t of Q' as illustrated lie on the extended segments of $\boldsymbol{AB}$ and $\boldsymbol{CA}$ and are therefore indeed negative. The closest point to Q' on the triangle in Figure 2.10(b) is A.

If the conditions above are not met, the closest point to Q' may lie on the edges of the triangle. As an example, the edge $\boldsymbol{AB}$ is analyzed in the following. Let r be the counterpart of s on $\boldsymbol{AB}$ from the vertex B. While s is slightly negative in Figure 2.10(b), r is positive, starting from B through A to the end of s. r is computed with

$$r = \frac{\boldsymbol{BQ} \cdot \boldsymbol{BA}}{\boldsymbol{AQ} \cdot \boldsymbol{AB} + \boldsymbol{BQ} \cdot \boldsymbol{BA}}\ . \tag{2.11}$$

To determine, whether a projected point Q' lies in the Voronoi feature region of $\boldsymbol{AB}$ with its closest point on the edge $\boldsymbol{AB}$, the following conditions must be met:

$$\begin{aligned} p_{QAB} &<= 0 \quad \text{and} \\ s &>= 0 \quad \text{and} \\ r &>= 0 \end{aligned} \tag{2.12}$$

p_{QAB} from Eq. 2.8 is negative, if Q' lies outside the edge $\boldsymbol{AB}$ and 0 if it is directly on the edge. At the same time, s and r are only positive (or 0) if Q' lies in the area between the vertex Voronoi regions of A and B. For example, S' fulfills these conditions as illustrated in Figure 2.10(b). Finally, the closest point to S' is a virtual point V on the edge $\boldsymbol{AB}$, which can be computed with

$$V = A + s * \boldsymbol{AB} \; . \tag{2.13}$$

Similarly, the same checks can be performed for the vertices B/C and the edges $\boldsymbol{BC}/\boldsymbol{CA}$.

The processes of finding the closest triangle (Phase 1) and finding the closest virtual point on this triangle (Phase 2) are repeated for all points of the query point cloud $\mathcal{P}_s$. The found virtual points and their corresponding query points then form point-to-mesh correspondences, which may be used in methods for local registration (Section 2.1.4.2).

In the area of BIM, the alignment of an accumulated laser scan to a CAD model is a common procedure for construction progress monitoring. Bosché et al. [48] proposed a semi-automatic registration with a planar surface feature extraction from the CAD model and the accumulated scan. A coarse alignment is achieved by manual point selection to create correspondences and ICP is employed for fine alignment.

2.1.5 Trajectory evaluation

The trajectory evaluation is a very important part of a scientific SLAM pipeline. Unfortunately, its importance is often underestimated and it generally receives little attention in publications. The major outputs of a SLAM pipeline are the map and the path a robotic platform has followed during its exploration, also called trajectory. It is composed of a sequence of poses, which are computed at discrete time steps from the sensor data, e.g., LiDAR/IMU. The output of a 3D LiDAR-SLAM algorithm is usually 6-DoF poses, containing the $[x, y, z]$ position and $[roll, pitch, yaw]$ Euler angles as orientation information. Alternatively, the latter can also be represented as a quaternion.

The evaluation of a trajectory output of a SLAM algorithm is conducted in two phases:

1. Alignment of a SLAM trajectory with a ground-truth (GT) trajectory
2. Trajectory error evaluation with selected metrics

Mistakes in the trajectory evaluation can therefore occur during the alignment of the two trajectories, but also when computing the error metric. In the following, techniques for the alignment of trajectories and selected evaluation metrics are discussed in more detail.

2.1.5.1 Trajectory alignment

The alignment of the SLAM and GT trajectories is only required, if they are not in a common coordinate frame, i.e., the transformation between the SLAM and GT coordinate systems is unknown. In a simulation environment, the real (GT) position of the LiDAR sensor and all other objects and their coordinate frames are known, even with a high update rate. Therefore, the trajectory computed by the SLAM algorithm can be directly compared to the GT trajectory (step 2) without an additional alignment.

Unfortunately, in reality there is a disconnect between the coordinate frames of the SLAM algorithm and the GT system. The coordinate frame of the map (e.g., world $\{W\}$) is commonly initialized at the starting position of the robot, while the coordinate frame of the GT system may have its own calibrated world frame as is the case for the motion capture infrared camera system "Vicon"[2]. Due to the statically mounted cameras, its coordinate frame is initialized once the calibration of the system has finished. All following pose recordings are then automatically in this frame. Since the robot does not always start in the same (exact) position due to positioning imprecision by the operator, the transform between the Vicon and SLAM coordinate frames is not only unknown but also varies for each exploration run. One could mark the starting location of the robot on the ground and compute the transform between the frames once, assuming always the same offset. However, robot positioning imprecision, even if it is a few centimeters in translation or a few degrees in rotation may result in an artificially high SLAM trajectory error. If undetected, wrong conclusions from the performance of the SLAM algorithm are drawn in the worst case.

Depending on the information retrieved from the GT system (i.e., 3-DoF or 6-DoF), different alignment strategies can be applied. The methods presented in the following assume the alignment of a SLAM and GT trajectory. However, they can also be used to determine the offset between two SLAM trajectories, e.g., different SLAM algorithms or parametrization.

Origin alignment One of the most common methods is the origin alignment strategy. Essentially, it determines the initial offset in translation and rotation between the first matching timestamp of the GT measurements and SLAM poses. The first common timestamp is assumed to be the start of the robot exploration and trajectory. If the GT system provides a full 6-DoF pose, the origin alignment is the most suitable strategy and results in the most accurate trajectory alignment.

Figures 2.11(a) and 2.11(b) show a 3D LiDAR-SLAM and GT trajectory before the alignment. The offset between the trajectories consisting of a translation and rotation is clearly visible. The rotational part includes an offset in yaw and pitch. Both trajectories consist of individual 6-DoF poses. The first measurements are illustrated as coordinate systems in the figures. The transform offset ${}^{\text{SLAM},1}\boldsymbol{T}_{\text{GT},1}$ can then be easily determined with

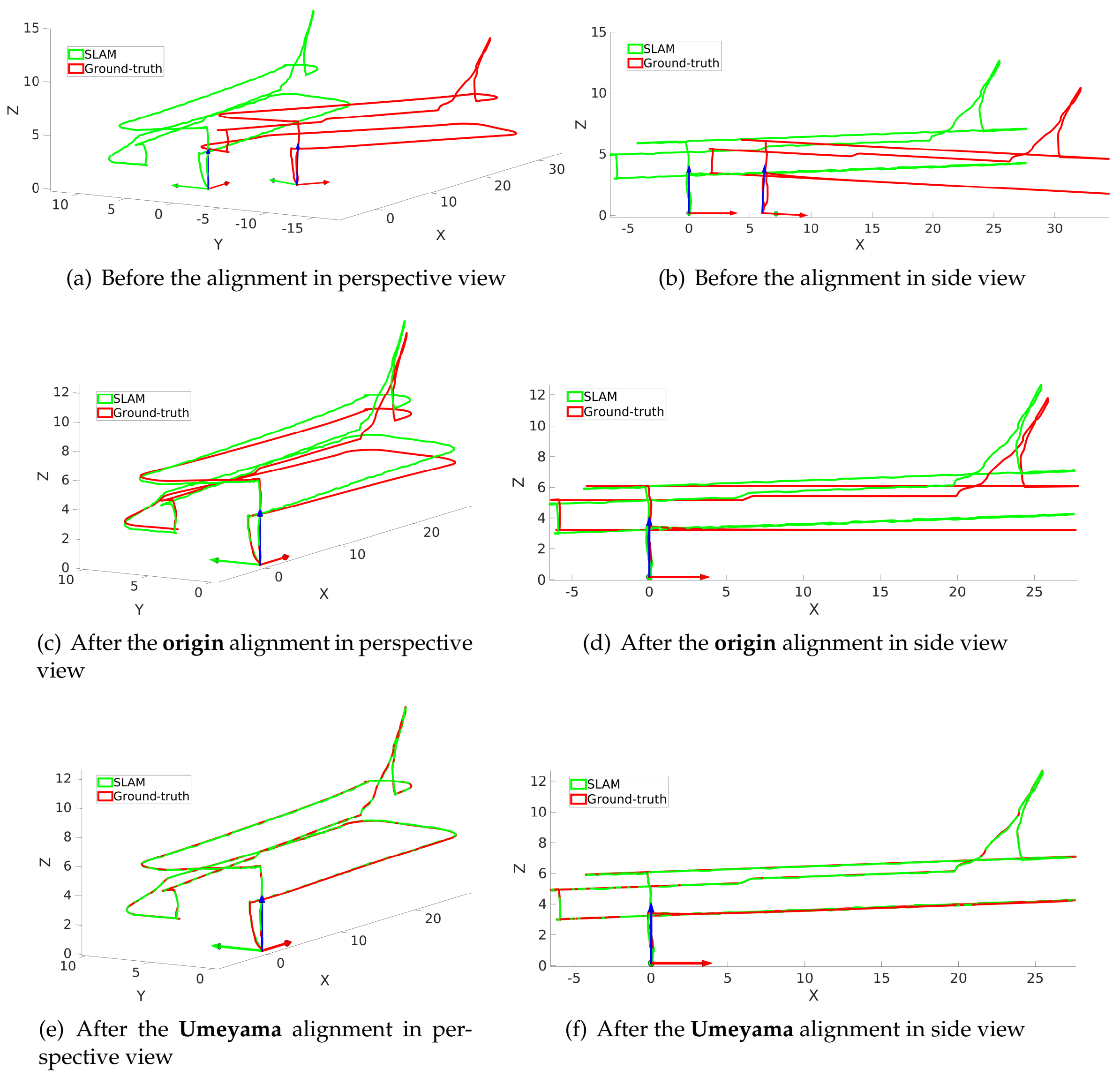
$$ {}^{\text{SLAM},1}\boldsymbol{T}_{\text{GT},1} = \boldsymbol{T}^{-1}_{\text{SLAM},1}\boldsymbol{T}_{\text{GT},1} \,, \tag{2.14} $$

where $\boldsymbol{T}_{\text{SLAM},1}$ is the first SLAM pose and $\boldsymbol{T}_{\text{GT},1}$ is the first GT measurement with corresponding matching timestamps. ${}^{\text{SLAM},1}\boldsymbol{T}_{\text{GT},1}$ can then be used to align the GT to the SLAM trajectory with

$$ \boldsymbol{T}_{\text{GT}_{\text{aligned}},1:n} = \boldsymbol{T}_{\text{GT},1:n}{}^{\text{SLAM},1}\boldsymbol{T}^{-1}_{\text{GT},1} \tag{2.15} $$

or vice versa. The GT trajectory consists of n 6-DoF poses, denoted as $\boldsymbol{T}_{\text{GT},1:n}$. It shall be noted that for the following error evaluation, it is not important in which direction the trajectories are aligned, i.e., the SLAM to the GT or vice versa. Usually, only the errors, e.g., Euclidean distances between the poses of the trajectories are computed. Figures 2.11(c) and 2.11(d) show the trajectories after the origin alignment. The coordinate systems now perfectly align. However, a height drift of the SLAM trajectory is clearly visible compared to the GT. Since the coordinate systems are aligned, the error is solely due to inaccurate pose esti-

[2] https://www.vicon.com



(a) Before the alignment in perspective view

(b) Before the alignment in side view

(c) After the **origin** alignment in perspective view

(d) After the **origin** alignment in side view

(e) After the **Umeyama** alignment in perspective view

(f) After the **Umeyama** alignment in side view

Figure 2.11: Trajectories of a 3D LiDAR-SLAM algorithm (green) and GT system (red) before and after different alignment strategies in perspective (left) and side views (right). The coordinate systems illustrate the first measurement values of the corresponding trajectories.

mation of the SLAM algorithm. After the origin alignment, the pose error can be determined with selected methods as discussed in Section 2.1.5.2.

Umeyama alignment Another common method for trajectory alignment is based on Umeyama's point set registration algorithm [49]. If no knowledge about the relative transformation between the SLAM and GT trajectories exists and no assumptions can be made, the method of Umeyama is a good choice for trajectory alignment. It finds the transformation $\boldsymbol{T}$ to align two point sets, e.g., X and Y. The point sets may also be the positional components $[x, y, z]$ of two trajectories $\boldsymbol{T}_{\mathrm{SLAM},1:n}$ and $\boldsymbol{T}_{\mathrm{GT},1:n}$, consisting of n individual poses with matching timestamps. The method is based on least-squares optimization using Singular

Value Decomposition (SVD) and it is capable of not only determining the rotation and translation but also the scale to align X and Y. Given the two point sets $X = \{x_1, x_2, \cdots, x_n\}$ and $Y = \{y_1, y_2, \cdots, y_n\}$, the rotation matrix $\boldsymbol{R}_{\text{Um}}$, translation vector $\boldsymbol{t}_{\text{Um}}$ and scale c to align the two sets are computed with [49]:

$$\mu_X = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2.16}$$

$$\mu_Y = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{2.17}$$

$$\sum_{XY} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \mu_Y)(x_i - \mu_X)^T \tag{2.18}$$

$$\boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T = \text{SVD}\left(\sum_{XY}\right) \tag{2.19}$$

$$\boldsymbol{S} = \begin{cases} \boldsymbol{I} & \text{if } \det(\boldsymbol{U})\det(\boldsymbol{V}) = 1 \\ \operatorname{diag}(1, 1, \cdots, 1, -1) & \text{if } \det(\boldsymbol{U})\det(\boldsymbol{V}) = -1 \end{cases} \tag{2.20}$$

$$\boldsymbol{R}_{\text{Um}} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T \tag{2.21}$$

$$\sigma_X^2 = \frac{1}{n}\sum_{i=1}^{n} ||x_i - \mu_X||_2 \tag{2.22}$$

$$c = \frac{1}{\sigma_X^2}\operatorname{tr}(\boldsymbol{D}\boldsymbol{S}) \tag{2.23}$$

$$\boldsymbol{t}_{\text{Um}} = \mu_Y - c\boldsymbol{R}\mu_X \tag{2.24}$$

μ_X and μ_Y are the centroids of X and Y, $\sum_{XY}$ is the covariance matrix, and $\boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T$ is the decomposed covariance. Equation 2.20 is the Kabsch algorithm to ensure a right-handed coordinate system when computing the rotation matrix $\boldsymbol{R}_{\text{Um}}$ in Eq. 2.21. σ_X^2 is the variance of X. If the scale c is not to be corrected, it can be set to $c = 1$ when computing the translation $\boldsymbol{t}_{\text{Um}}$ in Eq. 2.24.

Let

$$\boldsymbol{t}_{\text{GT},1:n} = \begin{pmatrix} \boldsymbol{t}_{\text{GT},1}^T & 1 \\ \vdots & \vdots \\ \boldsymbol{t}_{\text{GT},n}^T & 1 \end{pmatrix} \tag{2.25}$$

be the concatenation of the n translational components of the GT poses:

$$\boldsymbol{T}_{\text{GT},i} = \begin{bmatrix} \boldsymbol{R}_{\text{GT},i} & \boldsymbol{t}_{\text{GT},i} \\ 0 & 1 \end{bmatrix} \tag{2.26}$$

And let

$$
{}^{\mathrm{SLAM}}\boldsymbol{T}_{\mathrm{GT}} = \begin{bmatrix} \boldsymbol{R}_{\mathrm{Um}} & \boldsymbol{t}_{\mathrm{Um}} \\ 0 & 1 \end{bmatrix} \tag{2.27}
$$

be the transformation matrix consisting of the rotation matrix $\boldsymbol{R}_{\mathrm{Um}}$ and translational vector $\boldsymbol{t}_{\mathrm{Um}}$ computed with Umeyama's method in Equations 2.21 and 2.24, respectively. Note that in Eq. 2.26 $\boldsymbol{R}_{\mathrm{GT},i} = I$, if the GT provides only 3-DoF positions. The Umeyama-aligned GT positions $\boldsymbol{t}_{\mathrm{GT}_{\mathrm{aligned}},1:n}$ can then be computed with

$$
\boldsymbol{t}_{\mathrm{GT}_{\mathrm{aligned}},1:n} = \boldsymbol{t}_{\mathrm{GT},1:n}({}^{\mathrm{SLAM}}\boldsymbol{T}_{\mathrm{GT}}^{-1})^{T} \ . \tag{2.28}
$$

Alternatively, the GT poses can be aligned using the transformation matrices $\boldsymbol{T}_{\mathrm{GT},1:n}$ directly:

$$
\boldsymbol{T}_{\mathrm{GT}_{\mathrm{aligned}},1:n} = \boldsymbol{T}_{\mathrm{GT},1:n}{}^{\mathrm{SLAM}}\boldsymbol{T}_{\mathrm{GT}}^{-1} \tag{2.29}
$$

Figures 2.11(e) and 2.11(f) show the results of Umeyama's alignment method. Comparing it with the results of the origin alignment in Figures 2.11(c) and 2.11(d), it can be seen that the trajectories are better aligning with Umeyama due to the least-squares method. However, since the GT trajectory in this example provides full 6-DoF poses, the origin alignment is more accurate. In fact, Umeyama alignment also corrects the drift caused by the SLAM algorithm, which is still clearly visible in Figure 2.11(d). While the trajectory alignment with Umeyama seems successful visually, it significantly alters the following trajectory error computation and influences the conclusions drawn from the accuracy of the SLAM algorithm. The "evo_tools" software package[3] by Michael Grupp is a helpful tool to align trajectories, which also implements optional Umeyama alignment and scale correction. If not used with caution, the error displayed may be significantly lower than the true SLAM error.

If the GT system only provides 3-DoF position information, origin alignment can not be performed, since the rotation component of the relative transformation between the SLAM and GT trajectories is unknown. In this case, Umeyama's method can be used to determine parts of the rotation, i.e., roll, pitch, and/or yaw. A typical artifact of Umeyama alignment after the error computation is that the pose error does not start from 0, as one would expect. Instead, it may have already a higher value even at the first pose due to the least-squares optimization by aligning the whole trajectory. In contrast, the pose error when using origin alignment is 0 at the beginning since $\boldsymbol{T}_{\mathrm{SLAM},1} \equiv \boldsymbol{T}_{\mathrm{GT}_{\mathrm{aligned}},1}$.

A similar method to Umeyama is the rigid-body transformation estimation by Horn et al. [50].

Alignment using sensor information If the GT system provides only 3-DoF position data without orientation information, Umeyama alignment may be used to estimate the transformation between the trajectories. Instead of using Umeyama's method to estimate the full 6-DoF transformation, it can be used to only estimate some of the Euler angles, if some assumptions can be made. For example, if the XY-plane of the GT system is assumed to be

[3] https://github.com/MichaelGrupp/evo

fully parallel to the ground (i.e., $\text{roll}_{\text{GT},1} = 0$, $\text{pitch}_{\text{GT},1} = 0$), and an IMU sensor mounted on the LiDAR sensor measures $\text{roll}_{\text{SLAM},1}$ and $\text{pitch}_{\text{SLAM},1}$ with its accelerometer, then only the yaw offset needs to be determined by the Umeyama alignment. Hence, this removes the possibility to draw conclusions about the yaw drift of the SLAM trajectory but allows for proper evaluation of roll and pitch drift.

In special scenarios, LiDAR information can be used directly to determine the 6-DoF alignment transformation. If both, the position of a 3D object in the coordinate frame of the GT system and the geometry of the object in form of a 3D point cloud or mesh are known, local registration methods can be employed to align captured LiDAR data at the starting position of the robot to the known 3D model. Since the position of the known 3D object in the GT coordinate frame is known, the output transform of the local registration can be used to determine the trajectory alignment transformation ${}^{\text{SLAM}}\boldsymbol{T}_{\text{GT}}$.

2.1.5.2 Evaluation metrics

After the alignment of the SLAM and GT trajectories, the errors can be computed, which are the most important results to evaluate the performance of a SLAM algorithm. A direct influence on the evaluation metrics is the alignment strategy from Section 2.1.5.1. Wrong origin alignment will most likely increase the errors, while on the other hand, Umeyama alignment effectively removes drift and therefore artificially decreases the trajectory error. For this reason, the errors computed with the evaluation metrics are highly dependent on the proper trajectory alignment.

Evaluation metrics have the requirement of synchronized timestamps in common, i.e., the SLAM and GT system should be using the same master clock or should be synchronized by adjusting a time offset. Due to different sampling frequencies of the LiDAR sensor and GT system, timestamps will never match exactly. Instead, interpolation techniques on the poses of the trajectories can be used, e.g., linear interpolation. A maximum time interval threshold can be used to avoid too high interpolation errors if the time between two measurements is too large. For example, the interpolation between two SLAM poses with a time distance of $1\ s$ will lead to a much larger error compared to the interpolation between poses with a time distance of $0.1\ s$. If no maximum interpolation threshold is set, the errors may exceed the real position estimation errors of the SLAM system and influence the results.

A metric not further discussed in this thesis is the Relative Pose Error (RPE), which compares the relative drift of one trajectory compared to another. Kümmerle et al. [51] described a method to compare the relative poses of a trajectory with another. In other words, it computes the translational and rotational displacement between short sequences of the two trajectories and represents the offset in one single error metric.

In the following, the Absolute Pose Error (APE) and Rotational Error (RE) evaluation metrics are explained in more detail.

Absolute Pose Error (APE) It is one of the most used metrics when evaluating the performance of SLAM algorithms. The computation is identical to the Absolute Trajectory Error (ATE) used by Sturm et al. [52] for evaluating the performance of RGB-D SLAM. The authors provide an open-source implementation to compute the ATE in their TUM RGB-D benchmark suite[4].

The metric computes the translational offset between the absolute poses of two aligned trajectories. Assuming the comparison of aligned SLAM and GT trajectories, let

$$e_{\boldsymbol{t},i} = ||\boldsymbol{t}_{\text{SLAM},i} - \boldsymbol{t}_{\text{GT}_{\text{aligned}},i}||_2 \tag{2.30}$$

be the translational error between a SLAM and corresponding aligned GT pose (e.g., as in Eq. 2.28). The maximum APE of the SLAM trajectory can then be simply determined with

$$\text{APE}_{\text{max}} = \max(e_{\boldsymbol{t},1}, e_{\boldsymbol{t},2}, \cdots, e_{\boldsymbol{t},n}) \ . \tag{2.31}$$

Following the same principle, the median APE, mean APE, minimum APE and APE Root Mean Squared Error (RMSE) can be computed:

$$\text{APE}_{\text{median}} = \text{median}(e_{\boldsymbol{t},1}, e_{\boldsymbol{t},2}, \cdots, e_{\boldsymbol{t},n}) \tag{2.32}$$

$$\text{APE}_{\text{mean}} = \frac{1}{n}\sum_{i=1}^{n} e_{\boldsymbol{t},i} \tag{2.33}$$

$$\text{APE}_{\text{min}} = \min(e_{\boldsymbol{t},1}, e_{\boldsymbol{t},2}, \cdots, e_{\boldsymbol{t},n}) \ . \tag{2.34}$$

$$\text{APE}_{\text{RMSE}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} e_{\boldsymbol{t},i}^2} \tag{2.35}$$

APE_{min} is the least commonly used evaluation metric since the first SLAM and GT pose are perfectly aligned when using the origin alignment strategy. Therefore, $\text{APE}_{\text{min}} = 0$ and no conclusions can be drawn about the SLAM performance from this metric. A good combination is to provide result values for APE_{max}, $\text{APE}_{\text{median}}$, APE_{mean} and eventually APE_{RMSE}. Since the APE_{mean} is more susceptible to high APE_{max} outliers, the $\text{APE}_{\text{median}}$ is an important value for the performance evaluation of the SLAM algorithm.

Rotational Error (RE) Apart from the translational error computed with the APE metrics, the RE of the SLAM poses is another important metric when evaluating the performance of a SLAM algorithm. The RE can only be computed if both, the SLAM algorithm and GT system provide 6-DoF poses.

Assuming the SLAM and trajectory-aligned GT poses in homogeneous 4×4 format $\boldsymbol{T}_{\text{SLAM},1:n}$ and $\boldsymbol{T}_{\text{GT}_{\text{aligned}},1:n}$, the error transform between two matching poses with index i is then denoted as

$$^{\text{SLAM},i}\boldsymbol{T}_{\text{GT}_{\text{aligned}},i} = \boldsymbol{T}_{\text{SLAM},i}^{-1}\boldsymbol{T}_{\text{GT}_{\text{aligned}},i} \ . \tag{2.36}$$

Let $\boldsymbol{R}$ be the 3×3 rotation matrix component of an error transform $^{\text{SLAM},i}\boldsymbol{T}_{\text{GT}_{\text{aligned}},i}$. The error rotation matrix with a 'ZYX' axis order can then be converted to Euler angles (roll $= \phi$,

[4] https://vision.in.tum.de/data/datasets/rgbd-dataset/tools#evaluation

pitch $= \theta$, yaw $= \psi$) for easier interpretation:

$$s_y = \sqrt{\boldsymbol{R}_{1,1}^2 + \boldsymbol{R}_{2,1}^2} \tag{2.37}$$

$$\phi = \begin{cases} \arctan2(\boldsymbol{R}_{3,2}, \boldsymbol{R}_{3,3}) & \text{if } s_y \geq \varepsilon \\ \arctan2(-\boldsymbol{R}_{2,3}, \boldsymbol{R}_{2,2}) & \text{if } s_y < \varepsilon \end{cases} \tag{2.38}$$

$$\theta = \arctan2(-\boldsymbol{R}_{3,1}, s_y) \tag{2.39}$$

$$\psi = \begin{cases} \arctan2(\boldsymbol{R}_{2,1}, \boldsymbol{R}_{1,1}) & \text{if } s_y \geq \varepsilon \\ 0 & \text{if } s_y < \varepsilon \end{cases} \tag{2.40}$$

s_y is a variable to detect the phenomenon of angle singularities, which may lead to an effect often referred to as "gimbal lock". It causes the loss of one DoF, if two of the three axes are reaching a parallel configuration and is especially critical for mechanical gimbals. For this reason, the conditional terms are introduced to handle possible singularities properly. To express the rotational error in a single scalar, the Euclidean norm can be computed:

$$e_{\boldsymbol{R}} = \sqrt{\phi^2 + \theta^2 + \psi^2} \tag{2.41}$$

$e_{\boldsymbol{R}}$ is the rotational error of a single SLAM pose. Equations 2.31 to 2.35 for the APE computation can then also be applied using $e_{\boldsymbol{R}}$ to compute the metrics for the rotational error.

2.1.6 Extended Kalman Filter (EKF)

The EKF is a nonlinear state estimator often used to predict and correct the state of a system, which suffers from noise, e.g., measurement noise. Instead of fully relying on the system state computed from the most recent measurement, the EKF also considers the previous system state, covariances, and motion model. An easy example of an EKF application is the prediction of the position of an airplane, which is disappearing behind a cloud. Considering the motion before it disappears behind the cloud, the EKF is capable of predicting the position of the airplane for a short amount of time. Once the airplane reappears, the state of the EKF is corrected with the new observation.

In practice, EKFs are a widely used method for sensor fusion, e.g., late fusion. The 6-DoF pose output of a SLAM algorithm and 9-DoF IMU measurements (orientation, angular velocity, linear acceleration) can be used as input to an EKF. Since the SLAM output usually has a lower frequency than IMU sensors, the EKF can be used to effectively increase the update rate of the final fused pose output. Measurement covariance can be used to control the weight of the SLAM pose and IMU data, depending on the confidence in the current measurements.

Like the conventional Kalman Filter (KF) as its linear counterpart, it consists of a prediction and correction step. Note that in literature and implementations, the matrices may have different names, and also the formulation might be varying. The prediction step of an EKF can be denoted as follows:

$$\begin{aligned}\boldsymbol{x}_{K|K-1} =& f(\boldsymbol{x}_{K-1|K-1}) \\ \boldsymbol{P}_{K|K-1} =& \boldsymbol{F}_{K-1}\boldsymbol{P}_{K-1|K-1}\boldsymbol{F}_{K-1}^{T} + \boldsymbol{Q}_{K-1}\end{aligned} \tag{2.42}$$

$\boldsymbol{x}_{K|K-1}$ and $\boldsymbol{P}_{K|K-1}$ are the predicted state and predicted process noise covariance, respectively. $f(\cdot)$ is the state transition function. It is computed from the corrected previous system state $\boldsymbol{x}_{K-1|K-1}$. $\boldsymbol{F}_{K-1} = \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_{K-1|K-1}}$ is the Jacobian of the state transition function. $\boldsymbol{Q}_{K-1}$ is the constant process noise covariance, which is added after each prediction step. The values are tuned depending on the application. The correction step consists of the computation of the Kalman gain $\boldsymbol{K}_K$, system state $\boldsymbol{x}_{K|K}$ and state covariance $\boldsymbol{P}_{K|K}$.

$$\boldsymbol{K}_K = \frac{\boldsymbol{P}_{K|K-1}\boldsymbol{H}_K^T}{\boldsymbol{H}_K\boldsymbol{P}_{K|K-1}\boldsymbol{H}_K^T + \boldsymbol{R}_K} \tag{2.43}$$

$$\begin{aligned}\boldsymbol{x}_{K|K} =& \boldsymbol{x}_{K|K-1} + \boldsymbol{K}_K(\boldsymbol{z}_K - \boldsymbol{H}_K\boldsymbol{x}_{K|K-1}) \\ \boldsymbol{P}_{K|K} =& (\boldsymbol{I} - \boldsymbol{K}_K\boldsymbol{H}_K)\boldsymbol{P}_{K|K-1}(\boldsymbol{I} - \boldsymbol{K}_K\boldsymbol{H}_K)^T \\ &+ \boldsymbol{K}_K\boldsymbol{R}_K\boldsymbol{K}_K^T\end{aligned} \tag{2.44}$$

$\boldsymbol{H}_K$ is the measurement model matrix or is sometimes referred to as the "state-to-measurement" function. $\boldsymbol{R}_K$ is the measurement noise covariance, which specifies the confidence in the current measurement values. A higher covariance will decrease the weight of the Kalman gain $\boldsymbol{K}_K$. In Equation 2.44, the current state $\boldsymbol{x}_{K|K}$ and state covariance $\boldsymbol{P}_{K|K}$ are updated with the Kalman gain. During the state update, the Kalman gain is used to control the influence of the new measurement $\boldsymbol{z}_K$. If the Kalman gain is high, the new measurement is fully trusted and it therefore heavily influences the new system state. On the other hand, a low Kalman gain gives no weight to the new measurement and therefore only relies on the prediction from Equation 2.42. Finally, the state covariance $\boldsymbol{P}_{K|K}$ is also updated with the Kalman gain for the next EKF iterations.

Typically, an EKF runs with a periodic update function and a constant frequency. Incoming measurements are stored in chronologically-sorted queues. For each measurement in the queue, one prediction and correction step are performed. Although, in practice, situations may occur where the prediction step is omitted, i.e., when an old measurement is received. Care should be taken when setting the frequency of the EKF much higher than incoming odometry data, i.e., a SLAM algorithm has a high processing delay. Let's assume an EKF and IMU frequency of $100\,\mathrm{Hz}$ and a LiDAR-to-SLAM-pose delay of $50\,\mathrm{ms}$. Up to 5 IMU measurements will be integrated into the current state of the EKF until the SLAM pose from $50\,\mathrm{ms}$ old LiDAR data is available to the EKF. Since the integration of this old data into the state might have a significant negative effect on the reliability of the EKF output [53], the frequency of the EKF should not be set too high. Even though a high fused update rate may feel appealing. In an ideal case, the odometry measurement is instantaneous (e.g., from wheel encoders) and immediately available together with the IMU data to the EKF. To compensate for sensor delays, Das et al. [54] proposed the Augmented State EKF (AS-EKF) for real-time telepresence navigation using sensor measurements subject to transmission delays. Van Merwe et al. [55] proposed the use of a Sigma-Point Kalman Filter (SPKF) to fuse delayed GPS measurements with IMU and altimeter data.

2.2 Related work

This section gives an overview of LiDAR-SLAM algorithms, their differences, and approaches, in particular the LOAM framework [12]. Subsequently, publications are discussed, which incorporate prior knowledge of different kinds and formats in order to improve localization and mapping accuracy. Finally, this section presents related work in the area of trajectory and map optimization for LiDAR-SLAM by closing loops or absolute relocalization.

2.2.1 LiDAR-SLAM

LiDAR-SLAM algorithms have gained increasing attention from the robotics community since the first 2D LiDAR sensors became available. In the late years of the 1990s in modern LiDAR-SLAM, typically 2D LiDAR sensors were affordable and used to determine the pose in an unknown environment. In 1997, Lu et al. [56] presented a graph mapping approach for 2D scan matching by modeling poses as random variables and solving a maximum likelihood criterion.

If a 6-DoF pose and a 3D map representation were required, researchers mounted the sensors on actuators to continuously rotate the LiDAR sensor. Many different variations of actuated 2D LiDAR sensors for 3D perception were proposed. KaRoLa [57] is a hardware setup comprising a rotating 2D LiDAR, an embedded PC, and IMU including wireless connectivity. Bosse et al. [58] mounted a 2D LiDAR on a spinning platform and perform LiDAR sweep matching with ICP. Their approach is sometimes referred to as C-SLAM. Later they extended the approach by mounting a 2D LiDAR combined with an IMU on a spring [59]. They termed their innovative device *Zebedee*, which can be handheld, hands-free, or vehicle-mounted. Similarly, Schadler et al. [60] also use a continuously actuated 2D LiDAR sensor for 3D perception but use a surfel-based map representation. Surfels [61] are voxel-like volumes, containing surface shape and reflectance distributions of the points within the volume. By finding surfel associations between a scan and the target map, followed by a pose optimization using the Levenberg-Marquardt (LM) method, a 6-DoF pose can be determined. Since single 2D LiDAR scans are very sparse, Droeschel et al. [27] proposed to accumulate 2D scans from a rotating LiDAR with visual odometry from two stereo cameras. Obstacles and the robot position can then be extracted from the accumulated local map. Map points are stored in a multiresolution grid. This structure is also leveraged to find point correspondences for the pose estimation. Later, Droeschel et al. [62] proposed a multimodal sensor setup on a MAV, composed of a rotating 2D LiDAR, two stereo cameras, and ultrasonic sensors. Obstacle avoidance, trajectory planning, and pose estimation are performed with a multiresolution map. The ultrasonic sensors are able to detect windows, which is not reliably possible with a LiDAR or camera. In a later publication [63], the authors improved map maintenance for a more robust registration and demonstrated their approach on the DARPA Robotics Challenge (DRC) for human assistance in disaster scenarios. Kohlbrecher et al. [64] introduced Hector-SLAM, a popular 2D open-source SLAM framework. It is capable of integrating the pose estimation from scan-to-map matching with IMU measurements using a multiresolution grid map. It is available as an open-source software package in the Robot Operating System (ROS) framework [65]. One of the most known open-source frameworks for 2D LiDAR-SLAM, which supports loop closures, was introduced by Hess et al. [66] under the name "Google Cartographer". The map is divided into submaps. New scans are only inserted into the local submap, while other submaps are used for loop closure detection. For the scan matching, the authors employ a least-squares optimization approach. Cartographer with LiDAR-only sensor data is running in 2D by default, providing a 3-DoF pose and 2D map. It is also capable of running in 3D, computing 6-DoF poses. However, it requires additional IMU input to pre-transform 3D scans according to the orientation measured by the IMU.

Opromolla et al. [67] proposed a 2D LiDAR-Inertial SLAM framework, fusing IMU with the pose obtained from scan-to-scan matching using a modified ICP algorithm. Mapping is conducted with extracted line features from LiDAR scans. Kumar et al. [68] proposed to use two 2D LiDAR sensors and an IMU for UAV position estimation. The primary LiDAR is used together with IMU data to perform scan-to-scan matching, yielding a 2D position. Line segments are extracted from the secondary LiDAR data and used as 1D height information. The position and height are then fused in a Kalman Filter to obtain the 3D position.

The evaluation of the map quality is challenging if no GT poses are available. Typically, the accuracy of pose estimation is tightly coupled with the quality of the 3D reconstruction, since the estimated poses are used to accumulate the LiDAR scans in a common coordinate frame. Razlaw et al. [69] proposed the Mean Map Entropy (MME) and Mean Plane Variance (MPV) metrics to evaluate 3D map quality without a GT.

In recent years, prices of 3D LiDAR sensors dropped significantly and became more affordable. Besides the well-known 3D LiDAR manufacturer Velodyne, Ouster has recently started selling 3D LiDAR sensors with up to 128 scan lines at competitive prices. Hence, in the research community, there was a shift from 2D to 3D LiDAR sensors for 6-DoF pose estimation. SegMatch [70] is an open-source 3D LiDAR framework, which is based on extracting segments rather than point features from the LiDAR scan. An algorithm clusters points belonging to the same object and a feature vector is composed, describing statistics of the object/segment point cloud. A random forest classifier matches the segments between the source point cloud and a target map. The latter can be created online or pre-loaded from the previous exploration. The method also features a loop closing mechanism. The authors later further developed the approach in SegMap [71]. Instead of using hand-crafted descriptors to describe the segments, Convolutional Neural Networks (CNNs) are employed for descriptor vector generation, reconstruction, and segment classification, e.g., building, car, or other types. A major improvement is the capability to reconstruct the segments for obsta-

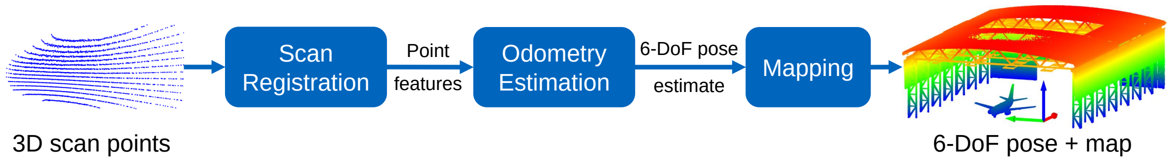


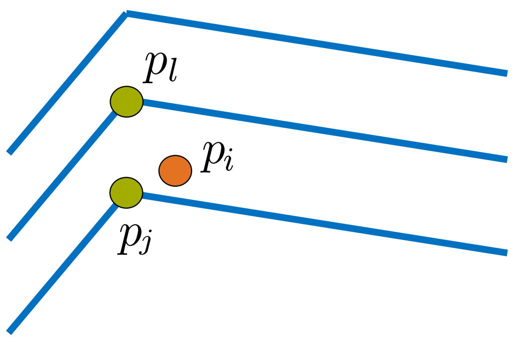
Figure 2.12: Pipeline of the LOAM framework by Zhang et al. [12]

cle avoidance and situation awareness by an operator. LOL [72], proposed by Rozenberszki and Majdik, is an open-source 3D LiDAR-SLAM framework. It uses segment features and correspondence estimation of SegMap [71]. The method requires an a priori known map for coarse global relocalization, e.g., from an offline recording that could be provided by automotive companies for autonomous driving scenarios. Fine relocalization is achieved with ICP. Zhen et al. [73] extended the Cartographer framework to support 2D and 3D static and rotating LiDAR sensors. They found that a rotating LiDAR is crucial for reliable height estimation. Despite multiple vertical scan lines, 3D LiDAR sensors typically only have a small vFoV, i.e., $30°$ for a Velodyne VLP-16 and $45°$ for an Ouster OS1-128. If the sensor is mounted on a UAV, not enough ground points are captured for reliable height estimation. Hence, the authors recommend additionally rotating the sensor to capture an even wider FoV. Dubé et al. [74] extended the SegMatch framework [70] later on, to support even multiple agents. Each agent creates its own source map, in which it localizes. Frequently, the source map is matched against a global target map to detect loop closures. The method uses the segment features proposed in the SegMatch approach for correspondence estimation. LLOAM [75] is a full 3D LiDAR-SLAM algorithm, similar to [74]. It performs real-time scan segmentation for odometry estimation, mapping, and loop closure.

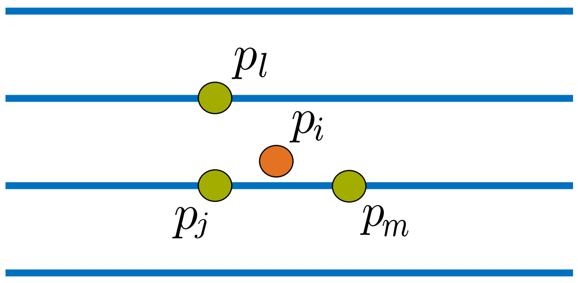
Zhang et al. [12] proposed the open-source LOAM framework based on corner and surface features extracted from 3D LiDAR scans. The contributions presented in this paper are all based on this framework, due to ease of implementation and top performance on the official KITTI odometry benchmark [76] for LiDAR-only sensor data as of today[5]. The framework is specialized in 3D LiDAR data with equally distributed vertical scan lines and a $360°$ hFoV. The pipeline can be seen in Figure 2.12. It basically consists of three modules: Scan registration, odometry estimation, and mapping. Their functionality is explained in more detail in the following.

Scan registration The name of the module may be misleading, since a scan registration may be understood as aligning the scan to another scan or point cloud. In fact, the module performs the feature extraction as described in Section 2.1.4.1. Incoming scan points are first assigned to the corresponding scan line they originated from. For this, the number of scan lines and vFoV of the LiDAR sensor must be known. Subsequently, the curvature of the points within each scan line is computed. A number of corner and surface features is extracted from six sections of the $360°$ LiDAR scan. The features are then sent to the odometry module.

[5] May 2022



(a) Correspondence detection for **corner** features. For a corner feature p_i in the current scan, p_j is the closest corner feature of the previous scan on any scan line and p_l is the closest corner feature to p_i on a neighboring scan line of p_j, i.e., above or below.



(b) Correspondence detection for **surface** features. For a surface feature p_i in the current scan, p_j is the closest surface feature of the previous scan on any scan line, p_m is the closest surface feature on either the same or neighboring scan line of p_j and p_l is the closest surface feature on definitely another scan line than p_j.

Figure 2.13: Correspondence detection for odometry estimation in the LOAM framework [12]. Blue lines depict the scan lines of the previous scan. p_i is the feature point in the current scan for which correspondences are found in the previous scan. Reproduced from [12].

Odometry estimation The main task of the odometry estimation module is to compute the transform between consecutive LiDAR scans $\mathcal{P}^{L_{t-1}}$ and $\mathcal{P}^{L_t}$ at times $t-1$ and t using the features from scan registration. As for any point cloud registration method, correspondences between the two scans need to be established, which are then used for an optimization step to obtain the transform.

Figure 2.13 depicts the correspondence detection between a feature p_i of the current scan and features of the previous scan. For *corner* features, a correspondence is created between p_i and two other corner features of the previous scan, as depicted in (a). Let $p_a = p_j$ and $p_b = p_l$, the residual vector $\boldsymbol{r}$ for a single corner feature p_i can then be computed with

$$\boldsymbol{r} = \frac{(\hat{p}_i - p_a) \times (\hat{p}_i - p_b)}{||(p_a - p_b||_2} . \tag{2.45}$$

$\hat{p}_i$ is the transformed corner feature of the current scan using the currently optimized transform. The optimization process aims at minimizing $||\boldsymbol{r}||_2^2$ by effectively converging the cross product in the numerator of the equation towards zero. By definition, a cross product is zero, when (1) either of the product's vectors point in the same or opposite directions, or (2) either of the vectors has a length of zero. In both cases, p_i would lie on the line defined by p_l and p_j, representing the edge in the LiDAR scan. In other words, the optimization process aims at minimizing the distance between $\hat{p}_i$ and the line defined by p_l and p_j.

For *surface* features, three correspondences are found on different scan lines in the previous scan as depicted in 2.13(b). p_j, p_l and p_m define a 3D plane. Let $p_a = p_j$, $p_b = p_l$ and $p_c = p_m$, the normalized (unit) normal vector $\boldsymbol{n}$ of the plane is then computed with

$$\boldsymbol{n} = \frac{(p_a - p_b) \times (p_a - p_c)}{||(p_a - p_b) \times (p_a - p_c)||_2} . \tag{2.46}$$

Finally, the scalar residual r for a single surface feature p_i can be computed with

$$r = (\hat{p}_i - p_a) \cdot \boldsymbol{n} \, . \tag{2.47}$$

The scalar product in Eq. 2.47 converges towards zero when the vectors become perpendicular to each other. A vector, which is perpendicular to the normal $\boldsymbol{n}$, lies on the plane defined by p_j, p_l and p_m. Hence, the optimization process aims at finding a transform, which creates a vector $\hat{p}_i - p_j$, lying on the plane. Effectively, it minimizes the distance between p_i and the planar surface.

Let $C_{\mathcal{E}} = \{c_{\mathcal{E}_1}, c_{\mathcal{E}_2}, \cdots\}$ and $C_{\mathcal{H}} = \{c_{\mathcal{H}_1}, c_{\mathcal{H}_2}, \cdots\}$ be the sets of correspondences for corner and surface features, respectively. The optimization can be written as a nonlinear least-squares problem with the total cost J:

$$J(\boldsymbol{q}, \boldsymbol{t}) = \sum_{c_{\mathcal{H}} \in C_{\mathcal{H}}} \rho(\|f_{\mathcal{H}}(c_{\mathcal{H}}, \boldsymbol{q}, \boldsymbol{t})\|_2^2) + \sum_{c_{\mathcal{E}} \in C_{\mathcal{E}}} \rho(\|f_{\mathcal{E}}(c_{\mathcal{E}}, \boldsymbol{q}, \boldsymbol{t})\|_2^2) \tag{2.48}$$

$f_{\mathcal{E}}(\cdot)$ and $f_{\mathcal{H}}(\cdot)$ are the cost functions computing the residuals from Eq. 2.45 and 2.47, respectively. $\| \cdot \|_2^2$ represents the squared L2-norm of the residuals, which is wrapped inside a Huber loss function $\rho(\cdot)$. The LOAM framework solves the optimization problem with the Levenberg-Marquardt trust-region algorithm [77], [78]. $\boldsymbol{q}$ and $\boldsymbol{t}$ are the quaternion and translational vectors of the optimized transform.

The correspondence estimation and optimization steps are repeated twice in the original implementation. The output of the odometry module is the transform between the previous and current LiDAR scan, denoted as ${}^{L_{t-1}}\boldsymbol{T}_{L_t}$, which can be created from $\boldsymbol{q}$ and $\boldsymbol{t}$ of Eq. 2.48 after the optimization. This transform serves as an initial estimate for the motion since the last LiDAR scan for the mapping module.

Mapping The mapping module is the most complex part of the LOAM algorithm [12]. The main task is to estimate a transform ${}^{W}\boldsymbol{T}_{L_t}$, which best integrates the current scan $\mathcal{P}^{L_t}$ into the (voxelized) map. With this optimized transform, the point features are then inserted into the map. In fact, LOAM keeps two separate maps for corner and surface features and jointly optimizes them. There is a deviation in the correspondence estimation and residual calculation between the original LOAM and the advanced A-LOAM[6] implementations, which is used in this thesis. Hence, the following mathematical notations are derived from the program code of the mapping module of A-LOAM.

Correspondences in the map are not restricted to scan lines, in contrast to the odometry estimation. First, a corner or surface feature $\hat{p}_i$ is transformed to the map frame after integrating the motion since the previous scan from the odometry module:

$${}^{W}\hat{\boldsymbol{T}}_{L_t} = {}^{W}\boldsymbol{T}_{L_{t-1}} {}^{L_{t-1}}\boldsymbol{T}_{L_t} \tag{2.49}$$

${}^{W}\boldsymbol{T}_{L_{t-1}}$ is the map-optimized transform and pose of the previous scan, ${}^{L_{t-1}}\boldsymbol{T}_{L_t}$ is the odometry estimate and ${}^{W}\hat{\boldsymbol{T}}_{L_t}$ is the initial guess of the current scan in the map frame after inte-

[6] https://github.com/HKUST-Aerial-Robotics/A-LOAM

grating the odometry estimate.

For the *corner* features, the five closest features in the map to a corner feature $\hat{p}_i$ in the current scan are determined with a nearest neighbor search. Let $v_1, \cdots, v_3$ and $\lambda_1, \cdots, \lambda_3$ be the eigenvectors and eigenvalues of the covariance matrix computed from the five corner points. A corner correspondence is created if $\lambda_3 > 3\lambda_2$, assuming an ascending order of eigenvalues. This ensures that the corresponding eigenvector v_3 is dominant and the 3D ellipsoid formed by the five closest corner features has a cigar-like shape, which could represent a wall corner or an edge. Let c be the centroid of the five features, then $p_a = 0.1v_3 + c$ and $p_b = -0.1v_3$. Eq. 2.45 can then be reused for the residual vector computation.

The five *surface* features closest to a surface feature $\hat{p}_i$ are found in the map with a *k*-nearest neighbor *k*-d tree search. Let $\mathcal{P}_{\text{NN}} = \{p_1, \cdots, p_5\}$ be the five nearest surface features to $\hat{p}_i$. Solving the linear system

$$\begin{pmatrix} -1 \\ \vdots \\ -1 \end{pmatrix} = \begin{pmatrix} p_{1,x} & p_{1,y} & p_{1,z} \\ \vdots & \ddots & \vdots \\ p_{5,x} & p_{5,y} & p_{5,z} \end{pmatrix} \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} \tag{2.50}$$

yields the normal vector $\boldsymbol{n}$ of the plane defined by $\mathcal{P}_{\text{NN}}$. In A-LOAM, the linear system is solved using QR decomposition with column pivoting for computational efficiency. For the following, let $\boldsymbol{n} = \frac{\boldsymbol{n}}{||\boldsymbol{n}||_2}$ be the normalized unit normal vector. If

$$\exists p_o \in \mathcal{P}_{\text{NN}} : |p_o \cdot \boldsymbol{n} + \frac{1}{||\boldsymbol{n}||_2}| > 0.2\,, \tag{2.51}$$

then $\hat{p}_i$ is rejected and will not form a correspondence. The equation essentially computes the distance of each of the five features $p_o \in \mathcal{P}_{\text{NN}}$ to the plane. This ensures the flatness and quality of the estimated planar surface. If the condition is false for all five features, the scalar residual r between the surface feature $\hat{p}_i$ and the plane can be computed with

$$r = \boldsymbol{n} \cdot \hat{p}_i + \frac{1}{||\boldsymbol{n}||_2}\,. \tag{2.52}$$

Once all corner and surface feature correspondences have been created, the optimization problem as in Eq. 2.48 can be reused also for the map optimization. Just as for the odometry estimation, the correspondence estimation and optimization are repeated twice by default. The output of the mapping module is the map-optimized pose ${}^{W}\boldsymbol{T}_{L_t}$, which is the best transform estimate to fit the LiDAR scan $\mathcal{P}^{L_t}$ into the map. At the same time, it is also the best pose estimate in the map frame for this scan. All LiDAR scans can be accumulated with the corresponding map-optimized transforms to create a dense 3D reconstruction as described in Section 2.1.1 and as illustrated in Fig. 2.2(a).

Since the publication of the LOAM algorithm's source code, several variants have been published. LeGO-LOAM [79] leverages the knowledge about a ground plane in the lower scan lines. The authors specify that the algorithm can also be used for air-born robots, but some modifications are needed. Recently, F-LOAM [80] with an even more efficient open-source implementation was published.

2.2.2 Leveraging prior knowledge for pose estimation and SLAM

Prior knowledge can be interpreted in many different ways. It may be, for example, a previously created map from a prior exploration with mapping. The main challenge, in this case, is the time that passes between the capture and reuse of the built map. Especially in outdoor environments, techniques for long-term localization and mapping become crucial. Seasons change the outlook of the environment significantly throughout the year, e.g., vegetation changes color from green to brown or yellow, flowers in spring flourish, and the winter brings snow and ice to the streets, coloring them white. But also buildings may change their appearance, get constructed or destructed at any time of the year. Movable objects such as pedestrians, bicyclists, or vehicles make the scenery dynamic.

2.2.2.1 LiDAR-SLAM with a previously created map

Meyer-Delius et al. [81] reuse an a priori static map for localization, but also build a temporary local map from 2D LiDAR scans. The temporary map is used to map objects, which are not present in the static map. Depending on whether the observations are more consistent with the static or temporary map, the corresponding map is chosen for localization. Parsley et al. [82] were one of the first to leverage prior knowledge for graph SLAM. They integrate landmarks from a previously known map as constraints into the graph structure. Walcott-Bryant et al. [83] utilize a dynamic pose graph with local submaps for low-dynamic environments. Changes in the 2D maps are detected and eventually updated. Tipaldi et al. [84] use a particle filter in combination with a hidden Markov model for lifelong localization. Their method uses an existing 2D map as input. They evaluate their approach on a dynamic parking lot with a static 2D LiDAR sensor. Biswas et al. [85] differentiate in their work between short-term and long-term features. The latter are static objects, e.g., walls, and short-term features are considered dynamic or movable objects. By extracting long-term features, a static map can be created to be reused in future explorations. Rosen et al. [86] proposed a persistence filter for semi-static environments. Each feature is assigned a persistence probability to tackle environmental evolution. In a sense, the features then evolve with the environment. Fehr et al. [87] use a pan-tilt RGB-D sensor to build a static map from several observation runs. Between the runs, the environment is changing and a change detection algorithm is capable of segmenting the dynamic objects. Shaik et al. [88] create a 2D offline map with Hector-SLAM [64] and extract only static parts, e.g., walls, in a static map. It is then used as prior knowledge for an online exploration, where the static map is fused with a newly created temporary map, containing the changes, for localization.

Dubé et al. [70], [71] proposed multiple approaches for 3D LiDAR sensors leveraging a target map, which can be created offline a priori or online for loop closure. Later [89], they proposed a Dynamic Voxel Grid (DVG) to store points for the local map, created online. The

DVG allows to only update voxels, which got new point insertions. The mapping module is based on SegMatch [70]. The tracking of feature segments allows for a robust map update of the target map. Egger et al. [90] proposed PoseMap. It is a 3D LiDAR-SLAM algorithm developed for lifelong localization, which mainly relies on a previously created map. 3D LiDAR data is generated from a continuously rotating VLP-16 to increase the FoV. The map can be updated and extended if significant changes are detected. The authors point out that only small extensions to the offline created map can be made since it does not feature global place recognition. Pfrunder et al. [91] create a map with a nodding 3D LiDAR by manually driving a vehicle with the sensor mounted in front. This offline-generated map is then used several months later as prior knowledge for the localization of an autonomous vehicle. The scan points are pre-transformed with a tightly coupled IMU mounted on the LiDAR. Finally, high-frequency wheel odometry is fused with the LiDAR-SLAM pose for a more robust localization. Chen et al. [92] proposed an approach using a DNN for overlap prediction between the current 3D scan and a pre-built map.

Indoors, related works also leverage a priori mapped environments in form of point clouds or floor plans as CAD models. Winterhalter et al. [93] use the RGB-D images and the depth information of a Google Tango device to find the pose on a previously known 2D floor plan. The pose is estimated with Monte-Carlo localization (MCL). Boniardi et al. [94] leverage an existing 2D floor plan from an architectural drawing to improve 2D LiDAR-SLAM. The floor plan is converted to a binary image with appropriate scale and resolution. It is then augmented with LiDAR scans to determine the pose of the robot and to map inconsistencies in the floor plan. Later, Boniardi et al. [95] improved this work by also handling dynamic environments. Mielle et al. [96] generalized this approach, by incorporating prior knowledge in form of a hand-sketched map into the matching process rather than using a highly accurate floor plan. They also proposed several works leveraging a 2D emergency plan for LiDAR-SLAM [97], [98].

2.2.2.2 3D object models for pose estimation

Rather than a previously known map, prior knowledge can also be just about the geometry of individual objects. If the poses of objects should be determined, prior knowledge about their geometry is beneficial. One application is the counting or bin picking of the same product on an assembly belt. Another reason to determine the pose of an object in a scene is to determine the distance of the sensor to the object, e.g., for heading estimation and collision avoidance. If the object is known to be static, the relative pose to the object can be leveraged to improve the ego-motion estimation.

Matching 3D models with point cloud data is not only relevant for robotic applications. Müller et al. [99] register two organ meshes for image-guided liver surgery. The models are created from Time-of-Flight (ToF) or CT data. For the registration, landmarks are usually manually selected. The proposed approach extracts local surface descriptors for coarse alignment, which is then refined with ICP. Bauer et al. [100] proposed a similar approach for the initial patient pose detection for radiation therapy. Another application is to find or track poses of rigid objects in a scene, e.g., using RGB-D image data. Armenise [101] described a method to find the exact 6-DoF position of an object in a 3D scene. The object is available as a CAD model and the scene is captured with a Kinect depth sensor. The method is termed the Iterative Closest Face (ICF) counterpart to ICP. In the work, simple CAD models and dense scene representation are used for the alignment. No BVH techniques or similar are employed, which raises the question of how the computational complexity scales when using complex CAD models with many faces. The ICF method was later improved by Ye et al. [102] using Particle Swarm Optimization (PSO). Wang et al. [103] proposed to extract circular and corner features from a CAD model and simulated depth data from a scene. The features are matched and optimized with ICP, giving the final pose of the CAD model in the scene. However, the features on the CAD model are required to be labeled and indexed beforehand. As an example, they use a truncated square pyramid to extract corner features and a vase and cup for the extraction of circular features. Dos Santos et al. [104] developed a method to track objects in RGB-D images given their known 3D models. Their tracking algorithm employs particle swarm optimization. The fitness score of the particles is computed from the Euclidean distances between the scene and 3D object, HSV color coordinates, and the angles of the normal vectors. The 3D model is used as a point cloud, which can either be sampled from a CAD model or be acquired from RGB-D sensors.

Also in the area of BIM, registering a CAD model to a measured point cloud finds many applications. One of the most common is the progress monitoring of construction sites. Kim et al. [105] proposed to convert the CAD model of a planned construction site to a uniformly sampled point cloud, before coarsely registering it to a measured point cloud with Principal Component Analysis (PCA). The Levenberg-Marquardt ICP (LM-ICP) [106] method is then used for fine registration. Héry et al. [107], [108] proposed works to estimate the ego-vehicle position relative to a vehicle driving in front. 2D LiDAR scans are either matched with the 2D point-sampled shape or with the polygon shape of the preceding vehicle, consisting of 2D vertices and edges. Interestingly, the authors came across similar scan-to-model alignment ambiguities as are faced in the contributions presented in this thesis. For example, the authors found that the relative pose estimation has a larger rotational error if only the back of the car is visible and not also the side of the car. Sandy et al. [109] developed the In situ Fabricator (IF), an autonomous robotic platform for building construction. It features an end-effector with brick-gripping capability and a 2D LiDAR sensor for positioning. In their experiments, the IF autonomously places bricks on staples (workpieces) at different places on a construction site. The robot pose is computed with 2D scan matching, while high-accuracy end-effector positioning is achieved by scan-to-model alignment. For this, the end-effector is moved to create a 3D representation of the workpiece and ground while the robot is stationary. A least-squares optimization of the accumulated scan to the simple CAD model, consisting of primitives, then determines the relative pose to the workpiece. This approach is similar to the ones discussed above, where the pose of an object is found by registering an accumulated or dense point cloud to a CAD model. Gawel et al. [110] extended Sandy's approach. They developed a mobile platform with an end-effector for drilling operations.

It is equipped with a 3D VLP-16 LiDAR and an IMU for SLAM. A prior map is generated by point-sampling a 3D CAD model of the construction site. High-accuracy localization is implemented by refining the 3D SLAM pose with scan-to-CAD registration. For this, several distance measurements with single-point Laser Range Finders (LRFs) on the end-effector are taken by moving it to different positions. The distances are then minimized together with the expected distances from the planes of the CAD model, i.e., orthogonal walls, obtained with ray tracing. This is necessary to perform highly precise drilling operations.

Kümmerle et al. [111] proposed a method, which leverages aerial images for 6-DoF pose estimation. 3D LiDAR scans obtained from a continuously actuated 2D LiDAR are converted to 2D range scans and matched to the Canny edges [112] computed from the aerial images. The pose estimate is then integrated into the optimization problem and jointly optimized.

2.2.3 Trajectory and map optimization for SLAM

Trajectory and map optimization are an important part of relocalization or loop closure methods. Relocalization is typically achieved by revisiting a place. Assuming that the revisited place is associated with an accurate absolute pose in a global frame, finding the relative transform to this pose by sensor data associations can yield an accurate absolute pose for relocalization. Afterward, a loop closure can be performed by correcting the past trajectory. This is often done using a pose graph. By adding graph constraints with high pose confidence, an optimization of the graph can be performed. This correction of the previous trajectory can be leveraged to correct the map of the LiDAR-SLAM algorithm.

In the following, the most recent 3D LiDAR-SLAM algorithms with loop closure are discussed. Liu et al. [113] leverage the existence of a ground plane and perform loop closure detection based on SegMatch [70]. Behley et al. proposed SuMa [114] for 3D LiDARs. Loop closures are robustly detected with surfel maps and virtual view rendering, even with low overlap. Chen et al. extended this work in SuMa++ [115] by semantic, point-wise labeling for dynamic object removal and improved scan registration using a Fully Convolutional Neural Network (FCN). Ćwian et al. proposed PlaneLOAM 2.0 [116], an algorithm based on LOAM [12] with advanced high-level geometric features for mapping and a loop closure detection module from SegMap [89]. It finds loop closures when compared to a global map and performs the optimization with added constraints to a factor graph. Chai et al. [117] proposed a framework for global relocalization. An offline map is created from a mesh. Online relocalization is performed with Locality Sensitive Hashing (LSH) and k-d trees. The authors claim to achieve an accuracy up to $20\,\mathrm{cm}$ at $10\,\mathrm{Hz}$. OverlapNet [118] converts the 3D LiDAR scans to a multi-channel 2D image. A neural network predicts the yaw angle delta between scans and an overlap score. The relative transform is refined with ICP.

Other methods for loop closures with the help of global cues exist. Relocalization can also take place by finding an accurate transform to an object or tag, which was extrinsically calibrated in the global frame, i.e., it has an associated accurate absolute pose. For example, SPM-SLAM [119] or TagSLAM [120] use geometrically calibrated markers for visual SLAM. The latter uses extrinsically calibrated AprilTags, which were distributed along a robot trajectory. If two or more markers in an image are detected, position tracking and loop closure can be performed.

2.3 Chapter summary

The first part of this chapter introduced background knowledge in the area of 3D LiDAR-SLAM. This was followed by methods for LiDAR data acquisition and different ways for point cloud and triangular mesh representation. In many LiDAR-SLAM and point cloud registration algorithms, features are extracted and correspondence distances minimized. As an example, the feature extraction process of the LOAM algorithm [12] was explained in detail. Subsequently, local registration methods for point cloud to point cloud alignment were explained, with a focus on the ICP algorithm. In some cases, the registration of a point cloud to a 3D triangular mesh is necessary. It was explained that BVHs can be used to efficiently find the closest triangle to a query point and how the AABB tree can be leveraged. Furthermore, the computation of the closest virtual point on a triangle to a query point was explained in detail. Techniques to align a GT and SLAM trajectory were discussed and commonly used evaluation metrics were described.

The second part of this chapter dealt with the discussion of state-of-the-art LiDAR-SLAM algorithms. Especially, the LOAM algorithm [12], which is extensively used in this thesis, was described in detail. Subsequently, related publications leveraging prior knowledge to improve pose estimation and LiDAR-SLAM performance were discussed. Finally, related works in the area of trajectory and map optimization specifically for relocalization and loop closure were elaborated.

Chapter 3

3D LiDAR odometry and mapping with a static self-generated initial map

This chapter introduces an approach to enable immutable initial map creation for 3D LiDAR SLAM prior to the exploration. Using simulated data, the proposed approach is benchmarked against state-of-the-art LOAM in three visual inspection scenarios.

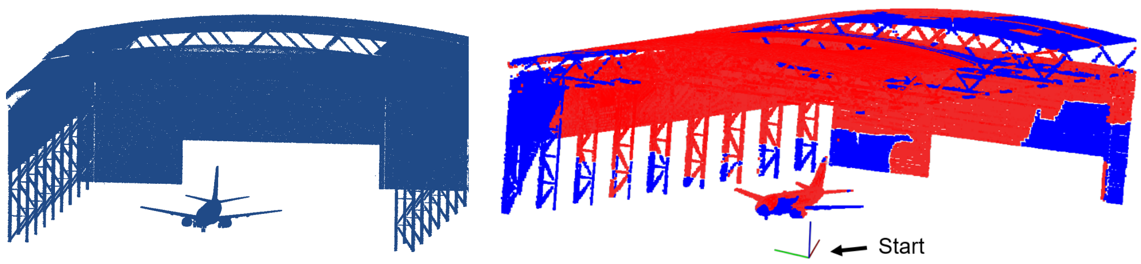
Parts of this chapter have been published in [5].

3.1 Problem statement

A major challenge for SLAM algorithms is the beginning of the exploration when the map has to be built from scratch. Visual-SLAM algorithms may even have to estimate the scale factor when creating the map. An initial wrong estimate can have a significant negative effect on the pose estimation and mapping for the whole exploration. LiDAR-SLAM algorithms do not have to estimate the scale due to the inherent range measurement, but first scan-to-scan registrations are crucial for the quality of the map and the accuracy of the pose estimation.

Also, in outdoor environments, maps can become very large and many parts may never be revisited, e.g., in an autonomous driving scenario through a city. However, in indoor scenarios with longer exploration periods inside the same hall or room many parts will be scanned much more often. The continuous map update process deteriorates the map quality over time but is required in dynamic environments to map new appearing obstacles, e.g., vehicles or people. Assuming a static environment, this may lead to unwanted effects on the map. Wrong pose estimations lead to incorrect LiDAR scan insertions into the map and may trigger the "double wall" effect, where the ground plane and walls grow thicker over time.

One type of prior knowledge, which is available to the robot before the start of the exploration is an initial map. It may be given to the robot, e.g., a previously built map from a separate exploration run, or it may be created from the starting position of the robot. Related works mainly focus on creating maps prior to the exploration for long-term localization and mapping as discussed in Section 2.2.2.1. In the LoLa-SLAM algorithm [3] an initial map is created by rotating a 3D LiDAR sensor at the starting position of the robot. LiDAR scans



(a) Conventional map (b) Immutable initial map (blue) and dynamic map (red)

Figure 3.1: Visualization of a conventional map (a) versus a map with initialization (b). The ground and parts of the hangar are removed for illustrative purposes.

are accumulated in a common coordinate frame and the scan points are inserted into the initial map. This map is then used as prior knowledge and extended during the exploration. The algorithm uses a voxelized map, so newly inserted scan points in occupied voxels are merged into the already existing voxel centroid. This diminishes the effect of slight pose estimation errors when inserting the new points, but still changes the position of the voxel centroids continuously.

The initial map can be created with much higher accuracy when using the transforms reported by the actuator compared to when using the transforms from the SLAM motion estimation. When the robot is still static, LiDAR scans can be transformed to a common coordinate frame with the actuator readings without motion estimation. In this chapter, a modification of the LOAM algorithm [12] is proposed, enabling immutable (fixed) initial map creation with a rotating 3D LiDAR sensor. Assuming a static indoor environment, the core idea is to keep the initial map unchanged (immutable) during the exploration and to only update the voxels, which were newly added since the start of the exploration. The proposed approach argues that updating the accurate initial map during the exploration will reduce the performance compared to leaving the initial map static since no changes are expected in the static environment. Fig. 3.1 shows a comparison between a conventional map (a) and a map with initialization (b). Blue points are accumulated during initialization and remain unchanged during exploration. They are part of the *initial* map. Red points are added during the exploration and are part of the *dynamic* map.

In the following sections, first, the methodology of the proposed approach is explained in detail. Second, the experimental setup comprised of the dataset generation is described and implementation details are given. Finally, extensive experimental results are presented before summarizing this chapter.

3.2 Methodology

The basic functionality of the LOAM algorithm [12] has already been described in the Sections 2.1.4.1 and 2.2.1. In the following, the notations $\mathcal{P}_{\mathcal{E}}$, $\mathcal{P}_{\mathcal{E}_{\text{less}}}$ represent the point clouds of sharp and less sharp corner features, and $\mathcal{P}_{\mathcal{H}}$, $\mathcal{P}_{\mathcal{H}_{\text{less}}}$ represent the point clouds of flat and less flat surface features, respectively.

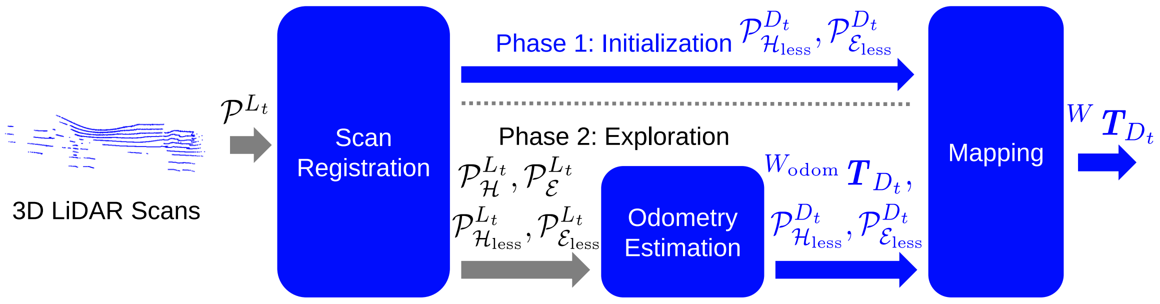

$\mathcal{P}^{L_t}$
3D LiDAR Scans
Scan Registration
Phase 1: Initialization $\mathcal{P}^{D_t}_{\mathcal{H}_\text{less}}, \mathcal{P}^{D_t}_{\mathcal{E}_\text{less}}$
Phase 2: Exploration
$\mathcal{P}^{L_t}_{\mathcal{H}}, \mathcal{P}^{L_t}_{\mathcal{E}}$
$\mathcal{P}^{L_t}_{\mathcal{H}_\text{less}}, \mathcal{P}^{L_t}_{\mathcal{E}_\text{less}}$
Odometry Estimation
${}^{W_\text{odom}}\boldsymbol{T}_{D_t}$,
$\mathcal{P}^{D_t}_{\mathcal{H}_\text{less}}, \mathcal{P}^{D_t}_{\mathcal{E}_\text{less}}$
Mapping
${}^{W}\boldsymbol{T}_{D_t}$

Figure 3.2: Overview of the proposed modifications to the LOAM framework to enable immutable initial map creation. Blue color indicates modification and gray/black means unchanged. Reproduced from [5], ©2021 IEEE.

Figure 3.2 shows an overview of the proposed modifications to the LOAM framework. Blue color indicates modifications and gray/black means unchanged. The original LOAM algorithm only features the exploration phase (Phase 2). All incoming LiDAR scans are used for motion estimation and all scan features are inserted into the map. To enable the initial map creation and bypass the motion estimation for the LiDAR scans acquired when the robot is still stationary, an initialization phase (Phase 1) is introduced to the pipeline. During this phase, the scan features are directly accumulated with the transforms reported by the actuator, forming the initial map of the environment in the mapping module. After the initialization, all scans are processed by the lower pipeline (Phase 2) and the motion is estimated before inserting the features into the map.

For the following algorithm description, the robot frame $\{D\}$ is introduced. The 3D LiDAR sensor is mounted via a gimbal/actuator on the robot. The forward kinematics of the LiDAR frame $\{L\}$ in the world frame $\{W\}$ at a time t can then be described as

$$ {}^{W}\boldsymbol{T}_{L_t} = {}^{W}\boldsymbol{T}_{D_t}\,{}^{D_t}\boldsymbol{T}_{L_t}\,. \tag{3.1} $$

${}^{W}\boldsymbol{T}_{D_t}$ is the estimated pose of the robot $\{D\}$ in $\{W\}$ obtained from the LOAM algorithm. ${}^{D_t}\boldsymbol{T}_{L_t}$ is the pose of the LiDAR sensor frame relatively to the robot frame. It is obtained from the actuator reading at time t. t is the timestamp of the LiDAR scan and is assigned at the driver level to the scan data.

3.2.1 Scan registration

Algorithm 1 shows the pseudo-code of the modified scan registration module. First, from an incoming LiDAR scan point cloud $\mathcal{P}^{L_t}$, the point features are extracted as described in Section 2.1.4.1. In fact, the term *extraction* might be a bit misleading, since it is rather a classification of the points into the four categories sharp ($\mathcal{P}_\mathcal{E}$), less sharp ($\mathcal{P}_{\mathcal{E}_\text{less}}$), flat ($\mathcal{P}_\mathcal{H}$) and less flat ($\mathcal{P}_{\mathcal{H}_\text{less}}$). In the A-LOAM implementation, $\mathcal{P}_{\mathcal{H}_\text{less}}$ are then downsampled with a voxel filter of fixed size $20\,\text{cm}$. The reason simply is that all scan points are assigned to be less flat if they do not belong to one of the other three categories. This significantly reduces the number of features to be processed and the computational complexity in the following steps. If the state is still in the initialization Phase 1, the actuator transform ${}^{D_t}\boldsymbol{T}_{L_t}$ is looked up for the

Algorithm 1: Scan registration algorithm. Modifications to the original algorithm are marked in blue color. Adapted from [5], ©2021 IEEE.

```
   Input: $\mathcal{P}^{L_t}, t$
   Output: $\mathcal{P}^{L_t}_{\mathcal{H}}, \mathcal{P}^{L_t}_{\mathcal{E}}, \mathcal{P}^{L_t}_{\mathcal{H}_{\text{less}}}, \mathcal{P}^{L_t}_{\mathcal{E}_{\text{less}}}, \mathcal{P}^{D_t}_{\mathcal{H}_{\text{less}}}, \mathcal{P}^{D_t}_{\mathcal{E}_{\text{less}}}$
1  foreach incoming scan point cloud $\mathcal{P}^{L_t}$ do
2      $\mathcal{P}^{L_t}_{\mathcal{H}}, \mathcal{P}^{L_t}_{\mathcal{E}}, \mathcal{P}^{L_t}_{\mathcal{H}_{\text{less}}}, \mathcal{P}^{L_t}_{\mathcal{E}_{\text{less}}} \leftarrow$ extractFeatures($\mathcal{P}^{L_t}$)
3      /* Downsample less flat points to $20\,\text{cm}$                    */
4      $\mathcal{P}^{L_t}_{\mathcal{H}_{\text{less}}} \leftarrow$ downsample($\mathcal{P}^{L_t}_{\mathcal{H}_{\text{less}}}$, 0.2)
5      if for initial map then
6          ${}^{D_t}\boldsymbol{T}_{L_t} \leftarrow$ readActuatorTransform($t$)
7          $\mathcal{P}^{D_t}_{\mathcal{H}_{\text{less}}}, \mathcal{P}^{D_t}_{\mathcal{E}_{\text{less}}} \leftarrow$ transform($\mathcal{P}^{L_t}_{\mathcal{H}_{\text{less}}}, \mathcal{P}^{L_t}_{\mathcal{E}_{\text{less}}}, {}^{D_t}\boldsymbol{T}_{L_t}$)
8          sendToMapping($\mathcal{P}^{D_t}_{\mathcal{H}_{\text{less}}}, \mathcal{P}^{D_t}_{\mathcal{E}_{\text{less}}}$)
9      else
10         sendToOdometry($\mathcal{P}^{L_t}_{\mathcal{H}}, \mathcal{P}^{L_t}_{\mathcal{E}}, \mathcal{P}^{L_t}_{\mathcal{H}_{\text{less}}}, \mathcal{P}^{L_t}_{\mathcal{E}_{\text{less}}}$)
```

time t and used to transform $\mathcal{P}^{L_t}_{\mathcal{H}_{\text{less}}}$ and $\mathcal{P}^{L_t}_{\mathcal{E}_{\text{less}}}$ to the robot frame $\{D\}$. Since the actuator and LiDAR are not synchronized, there is no exact actuator transform for the time t. Hence, a linear interpolation of the transform is performed for ${}^{D_t}\boldsymbol{T}_{L_t}$. The scan features in the robot frame are then sent directly to the mapping module. The reason why only less sharp and less flat features are used in the mapping module is that, as described in Section 2.1.4.1, they are more numerous and also include the sharp and flat features. If the state is already in Phase 2, all extracted scan features are sent to the odometry module.

3.2.2 Odometry estimation

The purpose of the odometry estimation module is to estimate real-time scan-to-scan motion, considering that the mapping process may not be real-time capable. Algorithm 2 shows the pseudo-code of the odometry module. The odometry optimization method is not modified by the proposed approach (lines 4-9). Here, the scan features from the scan registration module are first projected to the estimated state using the scan-to-scan transform ${}^{L_{t-1}}\boldsymbol{T}_{L_t}$ computed from the previous iteration. Feature correspondences are determined between the sharp/flat features of the current scan and the less sharp/less flat features of the previous scan, respectively. The correspondence distances are minimized in an optimization problem as described in Section 2.2.1. The process is repeated for two iterations by default. The actuator transform ${}^{D_t}\boldsymbol{T}_{L_t}$ is then looked up for the time t (line 12). The odometry module has its own world frame $\{W_{\text{odom}}\}$. The current position ${}^{W_{\text{odom}}}\boldsymbol{T}_{D_t}$ of the robot in the odometry world frame can then be computed with

$$ {}^{W_{\text{odom}}}\boldsymbol{T}_{D_t} = {}^{W_{\text{odom}}}\boldsymbol{T}_{D_{t-1}} {}^{D_{t-1}}\boldsymbol{T}_{L_{t-1}} {}^{L_{t-1}}\boldsymbol{T}_{L_t} {}^{D_t}\boldsymbol{T}^{-1}_{L_t} \,. \tag{3.2} $$

The equation allows us to perform the unmodified optimization of the LOAM algorithm and integrate the actuator transform afterward. The less sharp and less flat scan features are then

Algorithm 2: Odometry algorithm. Modifications to the original algorithm are marked in blue color. Reproduced from [5], ©2021 IEEE.

```
   Input: $\mathcal{P}_{\mathcal{H}}^{L_t}, \mathcal{P}_{\mathcal{E}}^{L_t}, \mathcal{P}_{\mathcal{H}_{\text{less}}}^{L_t}, \mathcal{P}_{\mathcal{E}_{\text{less}}}^{L_t}, t$
   Output: ${}^{W_{\text{odom}}}\boldsymbol{T}_{D_t}, \mathcal{P}_{\mathcal{H}_{\text{less}}}^{D_t}, \mathcal{P}_{\mathcal{E}_{\text{less}}}^{D_t}$
1  foreach incoming features $\mathcal{P}_{\mathcal{H}}^{L_t}, \mathcal{P}_{\mathcal{E}}^{L_t}, \mathcal{P}_{\mathcal{H}_{less}}^{L_t}, \mathcal{P}_{\mathcal{E}_{less}}^{L_t}$ do
2      if not the first scan then
3          /* Odometry optimization iterations */
4          for two iterations do
5              /* Pre-project the scan features of the current scan to the estimated
                  position using the transform from the last optimization */
6              $\hat{\mathcal{P}}_{\mathcal{H}}^{L_t}, \hat{\mathcal{P}}_{\mathcal{E}}^{L_t} \leftarrow$ transform($\mathcal{P}_{\mathcal{H}}^{L_t}, \mathcal{P}_{\mathcal{E}}^{L_t}, {}^{L_{t-1}}\boldsymbol{T}_{L_t}$)
7              $\mathcal{C}_{\mathcal{H}}, \mathcal{C}_{\mathcal{E}} \leftarrow$ computeCorrs($\mathcal{T}_{\mathcal{H}}^{L_{t-1}}, \hat{\mathcal{P}}_{\mathcal{H}}^{L_t}, \mathcal{T}_{\mathcal{E}}^{L_{t-1}}, \hat{\mathcal{P}}_{\mathcal{E}}^{L_t}$)
8              $\mathcal{O} \leftarrow$ createOptProblem($\mathcal{C}_{\mathcal{H}}, \mathcal{C}_{\mathcal{E}}, \mathcal{P}_{\mathcal{H}_{\text{less}}}^{L_{t-1}}, \mathcal{P}_{\mathcal{E}_{\text{less}}}^{L_{t-1}}, \bar{\mathcal{P}}_{\mathcal{H}}^{L_t}, \bar{\mathcal{P}}_{\mathcal{E}}^{L_t}$)
9              ${}^{L_{t-1}}\boldsymbol{T}_{L_t} \leftarrow$ solveOpt($\mathcal{O}$)
10     else
11         /* Initialize all transforms with identity */
12     ${}^{D_t}\boldsymbol{T}_{L_t} \leftarrow$ readActuatorTransform($t$)
13     /* Compute the transform from $\{D\}$ to $\{W_{\text{odom}}\}$ using Eq. 3.2 */
14     ${}^{W_{\text{odom}}}\boldsymbol{T}_{D_t} \leftarrow {}^{W_{\text{odom}}}\boldsymbol{T}_{D_{t-1}}\, {}^{D_{t-1}}\boldsymbol{T}_{L_{t-1}}\, {}^{L_{t-1}}\boldsymbol{T}_{L_t}\, {}^{D_t}\boldsymbol{T}_{L_t}^{-1}$
15     $\mathcal{P}_{\mathcal{H}_{\text{less}}}^{D_t}, \mathcal{P}_{\mathcal{E}_{\text{less}}}^{D_t} \leftarrow$ transform( $\mathcal{P}_{\mathcal{H}_{\text{less}}}^{L_t}, \mathcal{P}_{\mathcal{E}_{\text{less}}}^{L_t}, {}^{D_t}\boldsymbol{T}_{L_t}$)
16     sendToMapping($\mathcal{P}_{\mathcal{H}_{\text{less}}}^{D_t}, \mathcal{P}_{\mathcal{E}_{\text{less}}}^{D_t}, {}^{W_{\text{odom}}}\boldsymbol{T}_{D_t}$)
17     /* Build $k$-d trees for next scan */
18     $\mathcal{T}_{\mathcal{H}}^{L_t}, \mathcal{T}_{\mathcal{E}}^{L_t} \leftarrow$ buildKdTree($\mathcal{P}_{\mathcal{H}_{\text{less}}}^{L_t}, \mathcal{P}_{\mathcal{E}_{\text{less}}}^{L_t}$)
```

transformed to the robot frame with the actuator transform (line 15) and sent to the mapping module together with the optimized odometry pose (line 16). Finally, the k-d trees are built with the less flat and less sharp scan features in the LiDAR frame $\{L\}$. These are then reused for the correspondence computation in the next odometry estimation (line 7).

3.2.3 Mapping

Most modifications for the proposed approach are introduced in the mapping module of the LOAM algorithm. The core idea of the proposed approach is to keep the *initial* map and the *dynamic* map created during the exploration separate. In the following, the feature map created in Phase 2 will be called a *dynamic* map since it is continuously updated during the exploration. Alternatively, a new data structure could be implemented, which "freezes" the voxels of the initial map and only updates the voxels of the dynamic map. However, no such implementation exists. Algorithm 3 describes the modified mapping module schematically. First, the transform ${}^{W_{\text{odom}}}\boldsymbol{T}_{D_t}$ is integrated into the world frame of the map to obtain ${}^{W}\boldsymbol{T}_{D_t}$, which is the best odometry estimate. If the state is still in Phase 1, incoming scan features are directly added to the initial feature maps $\mathcal{P}_{\mathcal{H}_{\text{init}}}^{W}$ and $\mathcal{P}_{\mathcal{E}_{\text{init}}}^{W}$ (lines 4+5). This is possible since the robot is assumed to be stationary and the assumption $\{W\} \equiv \{D\}$ holds. If these are

Algorithm 3: Mapping algorithm with immutable initial map creation. Modifications to the original algorithm are marked in blue color. Adapted and modified from [5], ©2021 IEEE.

```
Input: ${}^{W}\boldsymbol{T}_{D_t} \leftarrow {}^{W_{\text{odom}}}\boldsymbol{T}_{D_t}, \mathcal{P}^{D_t}_{\mathcal{H}_{\text{less}}}, \mathcal{P}^{D_t}_{\mathcal{E}_{\text{less}}}, h, e$
Output: ${}^{W}\boldsymbol{T}_{D_t}$
foreach incoming features $\mathcal{P}^{D_t}_{\mathcal{H}_{less}}$ and $\mathcal{P}^{D_t}_{\mathcal{E}_{less}}$ do
    if for initial map then
        /* Robot is still static, so $\{W\} \equiv \{D\}$ */
        $\mathcal{P}^{W}_{\mathcal{H}_{\text{init}}} \leftarrow \mathcal{P}^{W}_{\mathcal{H}_{\text{init}}} + \mathcal{P}^{D_t}_{\mathcal{H}_{\text{less}}}$
        $\mathcal{P}^{W}_{\mathcal{E}_{\text{init}}} \leftarrow \mathcal{P}^{W}_{\mathcal{E}_{\text{init}}} + \mathcal{P}^{D_t}_{\mathcal{E}_{\text{less}}}$
        if last features for initial map then
            $\mathcal{P}^{W}_{\mathcal{H}_{\text{init}}}, \mathcal{P}^{W}_{\mathcal{E}_{\text{init}}} \leftarrow \text{downsample}(\mathcal{P}^{W}_{\mathcal{H}_{\text{init}}}, \mathcal{P}^{W}_{\mathcal{E}_{\text{init}}}, h, e)$
            $\mathcal{T}_{\mathcal{H}_{\text{init}}}, \mathcal{T}_{\mathcal{E}_{\text{init}}} \leftarrow \text{buildKdTree}(\mathcal{P}^{W}_{\mathcal{H}_{\text{init}}}, \mathcal{P}^{W}_{\mathcal{E}_{\text{init}}})$
        continue with next scan features
    $\overrightarrow{\mathcal{P}}^{W}_{\mathcal{H}_{\text{sub}}}, \overrightarrow{\mathcal{P}}^{W}_{\mathcal{E}_{\text{sub}}} \leftarrow \text{getSubmapCubes}(\overrightarrow{\mathcal{P}}^{W}_{\mathcal{H}_{\text{less}}}, \overrightarrow{\mathcal{P}}^{W}_{\mathcal{E}_{\text{less}}}, {}^{W}\boldsymbol{T}_{D_t})$
    $\mathcal{P}^{W}_{\mathcal{H}_{\text{sub}}}, \mathcal{P}^{W}_{\mathcal{E}_{\text{sub}}} \leftarrow \text{unifySubmapCubes}(\overrightarrow{\mathcal{P}}^{W}_{\mathcal{H}_{\text{sub}}}, \overrightarrow{\mathcal{P}}^{W}_{\mathcal{E}_{\text{sub}}})$
    /* Add initial map features to unified submaps */
    $\mathcal{P}^{W}_{\mathcal{H}_{\text{sub}}} \leftarrow \mathcal{P}^{W}_{\mathcal{H}_{\text{sub}}} + \mathcal{P}^{W}_{\mathcal{H}_{\text{init}}}$
    $\mathcal{P}^{W}_{\mathcal{E}_{\text{sub}}} \leftarrow \mathcal{P}^{W}_{\mathcal{E}_{\text{sub}}} + \mathcal{P}^{W}_{\mathcal{E}_{\text{init}}}$
    $\mathcal{P}^{D_t}_{\mathcal{H}_{\text{less}}}, \mathcal{P}^{D_t}_{\mathcal{E}_{\text{less}}} \leftarrow \text{downsample}(\mathcal{P}^{D_t}_{\mathcal{H}_{\text{less}}}, \mathcal{P}^{D_t}_{\mathcal{E}_{\text{less}}}, h, e)$
    $\mathcal{T}_{\mathcal{H}_{\text{sub}}}, \mathcal{T}_{\mathcal{E}_{\text{sub}}} \leftarrow \text{buildKdTree}(\mathcal{P}^{W}_{\mathcal{H}_{\text{sub}}}, \mathcal{P}^{W}_{\mathcal{E}_{\text{sub}}})$
    /* Map optimization iterations */
    for two iterations do
        $\mathcal{P}^{W_t}_{\mathcal{H}_{\text{less}}}, \mathcal{P}^{W_t}_{\mathcal{E}_{\text{less}}} \leftarrow \text{transform}(\mathcal{P}^{D_t}_{\mathcal{H}_{\text{less}}}, \mathcal{P}^{D_t}_{\mathcal{E}_{\text{less}}}, {}^{W}\boldsymbol{T}_{D_t})$
        $\mathcal{C}_{\mathcal{H}_{\text{less}}}, \mathcal{C}_{\mathcal{E}_{\text{less}}} \leftarrow \text{computeCorrs}(\mathcal{T}_{\mathcal{H}_{\text{sub}}}, \mathcal{P}^{W_t}_{\mathcal{H}_{\text{less}}}, \mathcal{T}_{\mathcal{E}_{\text{sub}}}, \mathcal{P}^{W_t}_{\mathcal{E}_{\text{less}}})$
        $\mathcal{O} \leftarrow \text{createOptProblem}(\mathcal{C}_{\mathcal{H}_{\text{less}}}, \mathcal{C}_{\mathcal{E}_{\text{less}}}, \mathcal{P}^{W}_{\mathcal{H}_{\text{sub}}}, \mathcal{P}^{W}_{\mathcal{E}_{\text{sub}}}, \mathcal{P}^{W_t}_{\mathcal{H}_{\text{less}}}, \mathcal{P}^{W_t}_{\mathcal{E}_{\text{less}}})$
        ${}^{W}\boldsymbol{T}_{D_t} \leftarrow \text{solveOpt}(\mathcal{O})$
    /* Add points to cube structure */
    foreach $p^{D_t}_{\mathcal{H}_{less}}$ in $\mathcal{P}^{D_t}_{\mathcal{H}_{less}}$ and $p^{D_t}_{\mathcal{E}_{less}}$ in $\mathcal{P}^{D_t}_{\mathcal{E}_{less}}$ do
        /* Transform point to map using newly optimized pose */
        $p^{W_t}_{\mathcal{H}_{\text{less}}}, p^{W_t}_{\mathcal{E}_{\text{less}}} \leftarrow \text{transform}(p^{D_t}_{\mathcal{H}_{\text{less}}}, p^{D_t}_{\mathcal{E}_{\text{less}}}, {}^{W}\boldsymbol{T}_{D_t})$
        /* Compute sq. distance to closest point in initial map */
        $d_{\mathcal{H}}, d_{\mathcal{E}} \leftarrow \text{nnSearch}(\mathcal{T}_{\mathcal{H}_{\text{init}}}, p^{W_t}_{\mathcal{H}_{\text{less}}}, \mathcal{T}_{\mathcal{E}_{\text{init}}}, p^{W_t}_{\mathcal{E}_{\text{less}}})$
        if $\sqrt{d_{\mathcal{H}}} < (h * \sqrt{3})$ then
            /* Too close to feature in initial map */
            continue with next point
        else
            $\overrightarrow{\mathcal{P}}^{W}_{\mathcal{H}_{\text{less}}} \leftarrow \overrightarrow{\mathcal{P}}^{W}_{\mathcal{H}_{\text{less}}} + p^{W_t}_{\mathcal{H}_{\text{less}}}$
        /* Repeat for $p^{W_t}_{\mathcal{E}_{\text{less}}}$ with $\sqrt{d_{\mathcal{E}}} < (e * \sqrt{3})$ */
    $\overrightarrow{\mathcal{P}}^{W}_{\mathcal{H}_{\text{less}}}, \overrightarrow{\mathcal{P}}^{W}_{\mathcal{E}_{\text{less}}} \leftarrow \text{downsample}(\overrightarrow{\mathcal{P}}^{W}_{\mathcal{H}_{\text{less}}}, \overrightarrow{\mathcal{P}}^{W}_{\mathcal{E}_{\text{less}}}, h, e)$
```

the last scan features for the initial maps, and the next scans enter Phase 2, the initial maps are voxelized (line 7). For this, the scalar parameters h and e define the voxel sizes of the surface and corner feature maps, respectively. Subsequently, the k-d trees are built (line 8) for efficient nearest neighbor search later on.

If the state is in Phase 2, the current best-estimate pose ${}^{W}\boldsymbol{T}_{D_t}$ is used to shift the submap window (line 10). The A-LOAM implementation uses a cube structure of size $50 \times 50 \times 50$ meters to efficiently store the features of the dynamic maps, denoted as $\overrightarrow{\mathcal{P}}^{W}_{\mathcal{H}_{\text{less}}}$ and $\overrightarrow{\mathcal{P}}^{W}_{\mathcal{E}_{\text{less}}}$. In total, the cubes cover an area of $250 \times 250 \times 250$ meters to limit the memory consumption and processing time. All the features in the cubes are then combined to the point clouds $\mathcal{P}^{W}_{\mathcal{H}_{\text{sub}}}$ and $\mathcal{P}^{W}_{\mathcal{E}_{\text{sub}}}$ (line 11). These contain all less flat and less sharp features of the dynamic map. Now, the scan features of the initial map are added (lines 13+14), complementing the dynamic map features. The current scan features are downsampled to match the resolution of the feature maps and k-d trees ($\mathcal{T}_{\mathcal{H}_{\text{sub}}}$, $\mathcal{T}_{\mathcal{E}_{\text{sub}}}$) are built. Two correspondence estimation and map optimization iterations are performed by the LOAM algorithm by default (lines 18-22). Here, the downsampled features of the current scan are transformed to the map frame with ${}^{W}\boldsymbol{T}_{D_t}$, which are then used to find correspondences with the nearest neighbor search using the previously built k-d trees. The correspondences are added to an optimization problem described in Section 2.2.1. Solving it yields the map-optimized pose ${}^{W}\boldsymbol{T}_{D_t}$, which is the best robot pose estimate $\{D\}$ in the world frame $\{W\}$ and subsequently used for the insertion of the current scan features into the dynamic maps.

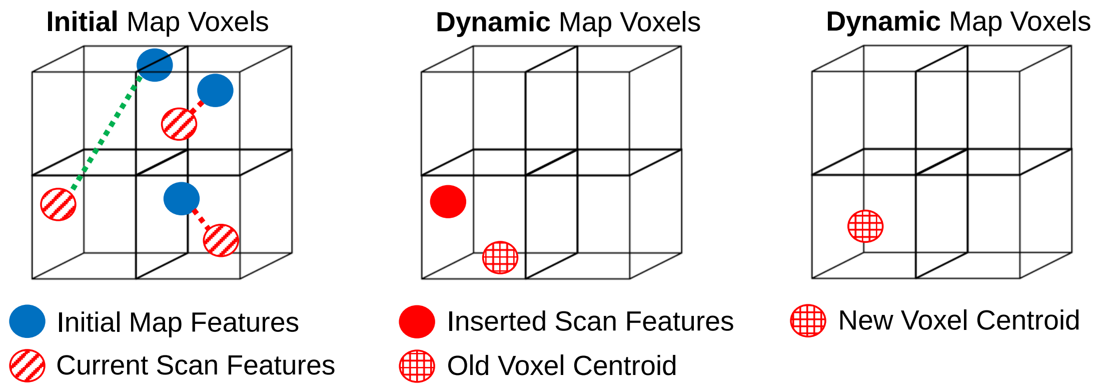
The current scan features are added to the maps in a point-by-point manner: A single feature point p is first transformed to the map frame with the map-optimized transform ${}^{W}\boldsymbol{T}_{D_t}$ (line 26). If the point is too close to a voxel centroid of the initial map, it is not added to the dynamic map. This ensures that the initial and dynamic maps are complementing each other and have no overlap. For this, the k-d trees built in Phase 1 are used for the nearest neighbor search to obtain the squared distance d (line 28). This process is illustrated in Fig. 3.3(a). As an example, a surface feature point $p^{W_t}_{\mathcal{H}_{\text{less}}}$ is not added to the dynamic map if $\sqrt{d_{\mathcal{H}}} < (h * \sqrt{3})$. In other words, a point is added to the corresponding cube in the dynamic map (line 33), if the Euclidean distance to the nearest point in the initial map is larger than the diagonal of the voxel as shown in (b). If it is closer, then this point is skipped. Finally, each cube of the dynamic map is downsampled. This effectively merges the new scan points into the voxel centroids of the dynamic map, see (c).

The unified initial and dynamic feature maps can be used for real-time visualization, but most importantly, the map-optimized transform ${}^{W}\boldsymbol{T}_{D_t}$ can be used for real-time navigation, e.g., for autonomous robots.

3.3 Experimental setup

The Gazebo simulation environment[1] is used to generate datasets for the evaluation. A quadcopter UAV is used as a robotic platform, see Figure 3.4. A virtual Velodyne VLP-16 LiDAR sensor is mounted on the UAV via a gimbal. The LiDAR sensor is continuously actuated

[1] http://gazebosim.org/

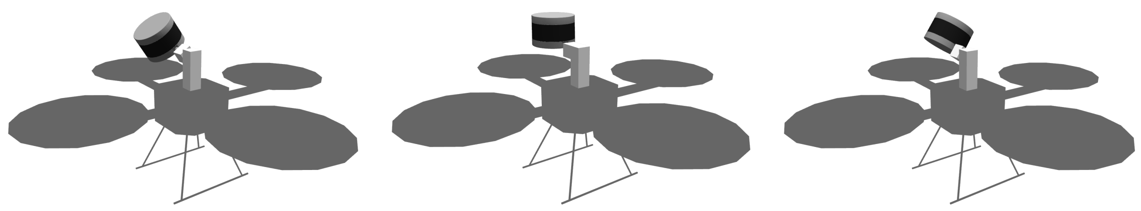


(a) Distance computation from a feature point of the current scan and points of the initial map (Alg. 3 line 28)

(b) Scan point insertion into the dynamic map (Alg. 3 line 33). The other voxels are empty since the initial map (a) has scan points in the corresponding voxels.

(c) The new scan point is merged with the old centroid in a voxelization step (Alg. 3 line 35).

Figure 3.3: Approach for scan insertion when using an immutable initial map.



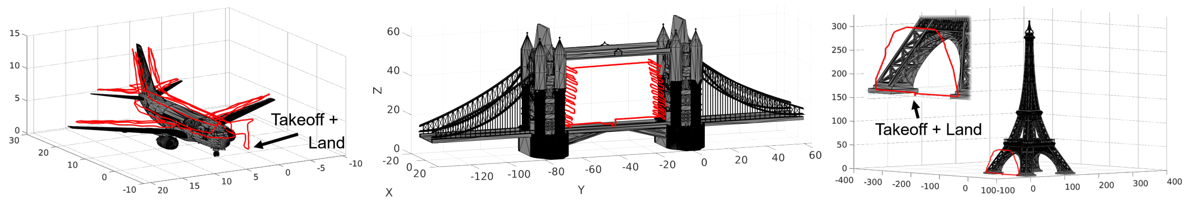
(a) $-0.6\,\mathrm{rad} \approx -34\,\mathrm{deg}$ (b) Initial LiDAR position (c) $+0.6\,\mathrm{rad} \approx +34\,\mathrm{deg}$

Figure 3.4: Illustration of the simulated quadcopter UAV with a VLP-16 LiDAR sensor mounted on a gimbal. The sensor is continuously actuated between $-0.6\,\mathrm{rad}$ (a) and $+0.6\,\mathrm{rad}$ (c) during the initialization (Phase 1) and during exploration (Phase 2).

back and forth between $\pm 0.6\,\mathrm{rad} \approx 34\,\mathrm{deg}$ during the map initialization (Phase 1), but also during the exploration (Phase 2). The virtual LiDAR follows the manufacturer specifications as described in Section 2.1.2. Acquired LiDAR scans are subject to zero-mean Gaussian noise with $\sigma = 0.03$ to simulate ranging errors along the ray direction, as in [31]. A LiDAR *sweep* is the set of scans acquired during one full actuation round. A sweep starts at the initial position (see Fig. 3.4(b)), turns to the right (c) and to the left (a), before ending the sweep by returning to the initial position. The next sweep then continues following the same actuation pattern.

Three datasets of visual inspection scenarios were generated. Figure 3.5 shows the scenarios and trajectories in red color. Dataset 1 is an indoor visual airplane inspection scenario inside a hangar and Datasets 2 and 3 are outdoor visual inspection scenarios at the Tower Bridge (b) and the Eiffel Tower (c), respectively. The triangular meshes are scaled to their real-world counterparts.

Table 3.1 shows the metadata about the three generated datasets. The average velocity of $0.5\,\mathrm{m/s}$ for all three datasets is due to the consistent autonomous control of the UAV in



(a) Dataset 1 - Indoors (b) Dataset 2 - Outdoors (c) Dataset 3 - Outdoors

Figure 3.5: Illustration of the three generated datasets. Dataset 1 (a) is indoors and Datasets 2 (b) and 3 (c) are outdoors. The hangar in (a) is not displayed for illustrative purposes. The trajectory followed by the simulated UAV is shown in red. Adapted from [5], ©2021 IEEE.

Table 3.1: Metadata about the three generated datasets of Fig. 3.5

	#scans	**length** (m)	**duration** (s)	**avg. vel** (m/s)	**max. vel.** (m/s)
Dataset 1	9802	475	980	0.48	1.21
Dataset 2	18605	933	1860	0.50	0.97
Dataset 3	5674	291	567	0.51	0.72

the simulation. It follows a predefined path using an autonomous flight controller. Dataset 2 is the largest with over $18k$ LiDAR scans and a path length of over $900\,\mathrm{m}$. The simulation allows for high-frequency, highly accurate GT poses and the generation of datasets, which would not be trivial to realize [31], i.e., Datasets 2 and 3.

The communication between the modules as shown in Fig. 3.2 is handled by ROS "melodic"[2]. The parameters of the LOAM algorithm are left as default.

3.4 Experimental results

The proposed approach creates an initial map by leveraging the known forward kinematics to the LiDAR sensor through the actuator transform while the robot is still stationary. The quality of the initial map is therefore dependent on the accuracy of the LiDAR measurements, i.e., ranging errors, and the accuracy of the actuator readings. Since the LOAM algorithm is using voxelized feature maps, the size of the voxels influences the representation of the underlying surface by the voxel centroid.

This section first presents an analysis, of how the actuator error as a function of the number of points (#points) within a single voxel influences the voxel centroid position relatively to the underlying surface. Subsequently, results are presented by means of RMSE for a whole triangular mesh and varying voxel size as a function of #sweeps. The voxel size and #sweep parameters for the initial map creation are derived and discussed in these experiments. The last section presents the results when using a static self-generated initial map before the exploration by means of APE and RE for all three datasets.

[2] http://wiki.ros.org/melodic

3.4.1 Effect of voxelization and measurement imprecision on surface representations

Essentially, LiDAR sensors create a sampled representation of a surface (real or simulated). Measured points are subject to ranging errors, reflections, and motion distortion. When the sensor is mounted on an actuator, the measured points are additionally subject to actuator reading imprecision of the current rotation angle. With the continuous actuation of the sensor, more points are acquired and a more dense and more complete representation of the surface is created. Wang et al. [121] and Anderson et al. [122] developed a *Lissajous* scan pattern and investigated the benefit for MEMS and 2D LiDAR sensors, respectively. Scan patterns can also be applied to 3D LiDAR sensors to further increase the FoV.

LiDAR-SLAM algorithms often need to operate in real-time to compute the current position in a built map. Hence, these algorithms do not keep all of the captured points but operate on voxelized maps to keep computational complexity and memory consumption tractable with increasing map sizes during robotic explorations. Different voxelization methods exist, such as replacing the measured points with the centroid of the voxel. While this is certainly the fastest way, it is not the most accurate. The goal of a voxel centroid is to represent the underlying surface in the best way. A common technique is to use the centroid of the points inside the voxel as the voxel centroid. Especially for voxels, which are only occupied in a corner, this method adapts much better to the real surface. However, deciding which point in a voxel best represents the underlying surface is not trivial.

3.4.1.1 Single voxel

This section investigates the effect of LiDAR ranging errors and single-axis actuator imprecision on the centroid position for a single voxel with a planar and curved surface. The centroid position of distorted points sampled on the surface is evaluated by (1) means of distance to the underlying surface and by (2) means of distance to the real centroid of the surface. The real centroid is computed from very densely sampled points without distortion.

A point $p = [x, y, z]$ is measured on a surface at a distance d from the LiDAR sensor at the position of origin o. p is distorted along the ray direction due to ranging errors of the sensor and is additionally subject to actuator reading imprecision with angle α. Assuming a zero-mean normal distribution, the ranging errors and actuator imprecision can be modeled as Gaussian random variables G and H, respectively:

$$\begin{aligned} G &\sim \mathcal{N}(0, 0.03^2) \\ H &\sim \mathcal{N}(0, 0.1^2) \end{aligned} \tag{3.3}$$

The standard deviation of $0.03\,\mathrm{m}$ follows the typical ranging errors of a Velodyne VLP-16 LiDAR sensor (see Section 2.1.2) and $0.1\,\mathrm{deg}$ is the imprecision of a $360°$ Dynamixel actuator with a resolution of $4096\,\mathrm{pulse/rev}$. Although, considering the empirical rule of statistics for normal distributions, these values are even conservative.

A distorted point $\hat{p}$ can then be described as

$$\begin{aligned} & \hat{p} = \boldsymbol{R} * \theta(p, g, o) \\ \text{with} \quad & \theta(p, g, o) = p * \left(1 - \frac{g}{||p-o||_2}\right) + o * \frac{g}{||p-o||_2} \,. \\ \text{For o = [0,0,0]:} \quad & \theta(p, g, o) = p * \left(1 - \frac{g}{||p||_2}\right) \end{aligned} \tag{3.4}$$

g is a ranging error value drawn from the distribution G (Eq. 3.3). The function θ distorts the point p along the ray direction, emitted from the LiDAR sensor at position o. $\boldsymbol{R}$ is a 3×3 rotation matrix $\boldsymbol{R} = \boldsymbol{R}_{\text{yaw}} \boldsymbol{R}_{\text{pitch}} \boldsymbol{R}_{\text{roll}}$ with a 'ZYX' axis order. In this example, p is distorted around the pitch axis by an angle α drawn from the distribution H (Eq. 3.3). $\boldsymbol{R}$ is then computed with

$$\boldsymbol{R} = \begin{bmatrix} \cos(\alpha\frac{\pi}{180}) & 0 & \sin(\alpha\frac{\pi}{180}) \\ 0 & 1 & 0 \\ -\sin(\alpha\frac{\pi}{180}) & 0 & \cos(\alpha\frac{\pi}{180}) \end{bmatrix} . \tag{3.5}$$

The centroid $\bar{p}$ of n points p can be computed as

$$\bar{p} = \frac{1}{n} \sum_{i=1}^{n} p_i \,. \tag{3.6}$$

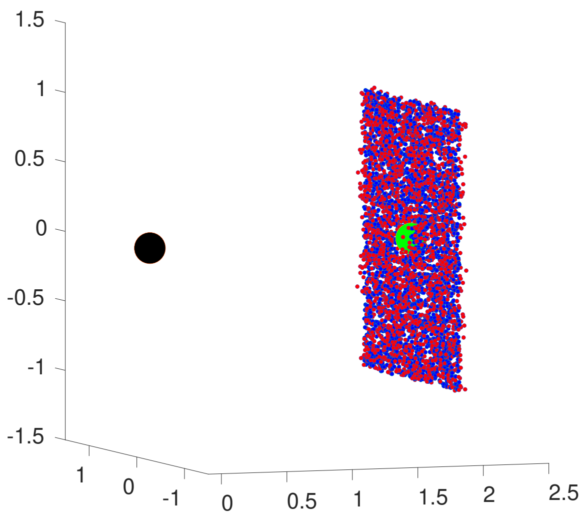
In the following, an investigation is conducted on how the measurement imprecision mentioned above influences the centroid position, considering different distances d and the number of points on a surface inside the voxel. Three types of centroids will be differentiated:

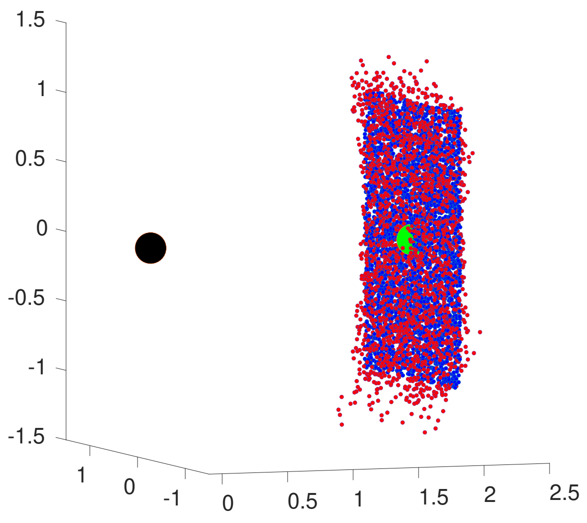
1. $\bar{p}$ - centroid of a number of undistorted points sampled on the surface
2. $\bar{p}_D$ - centroid of the points in (1) after distortion
3. $\bar{p}_S$ - real centroid of the surface (e.g., centroid of infinitely sampled surface)

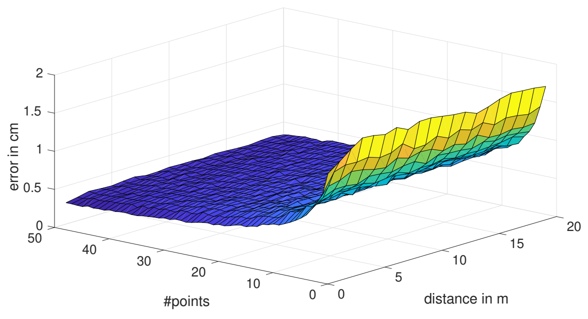
The centroid of *perfectly* sampled points on the surface is denoted as $\bar{p}$, the centroid of the corresponding *distorted* points is denoted as $\bar{p}_D$ and the centroid of the *surface* is denoted as $\bar{p}_S$.

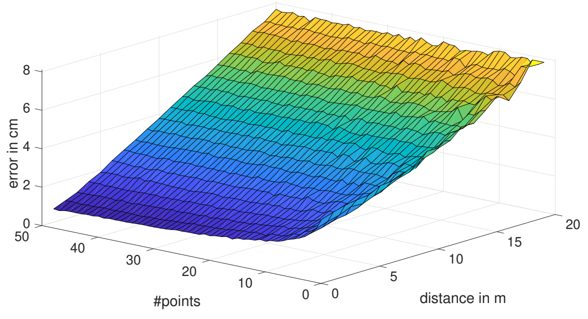
Planar surface

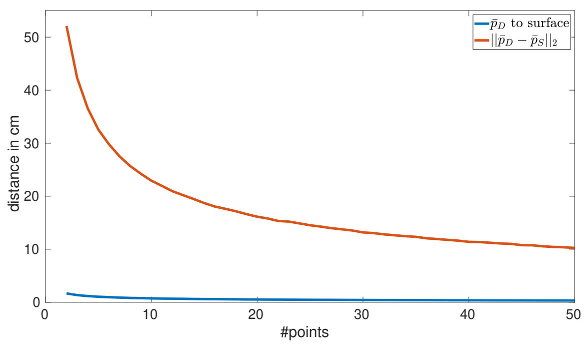
A number n of points $\mathcal{P} = \{p_1, p_2, \cdots, p_n\}$ is uniformly sampled on a planar surface, e.g., a wall segment for $x = d$, $y = [-v, +v]$ and $z = [-v, +v]$ with $v = 1\,\text{m}$ corresponding to a voxel size of $200\,\text{cm}$. The surface forms a square on a 3D hyperplane. The sampled points $\mathcal{P}$ are then distorted according to Eq. 3.4 with random distortions drawn from the distributions in Eq. 3.3 and also $H \sim \mathcal{N}(0, 5.0^2)$ with an increased actuator imprecision for illustrative purposes.


(a) 3000 points with actuator $\sigma = 0.1$


(b) 3000 points with actuator $\sigma = 5.0$


(c) Distance of $\bar{p}_D$ to the surface for actuator $\sigma = 0.1$


(d) Distance of $\bar{p}_D$ to the surface for actuator $\sigma = 5.0$


(e) Distance of $\bar{p}_D$ to the surface and $\bar{p}_D$ to $\bar{p}_S$ for actuator $\sigma = 0.1$

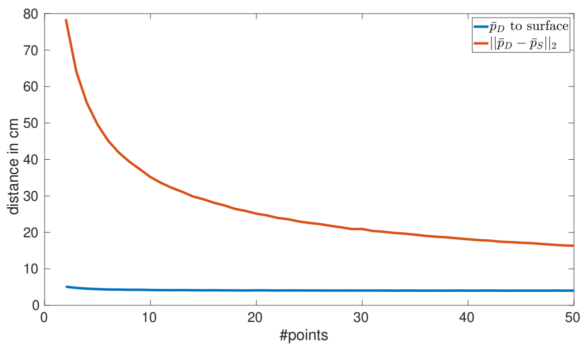

(f) Distance of $\bar{p}_D$ to the surface and $\bar{p}_D$ to $\bar{p}_S$ for actuator $\sigma = 5.0$

Figure 3.6: Random uniform generation of point samples on a wall segment with actuator $\sigma = 0.1$ (a) and $\sigma = 5.0$ (b). The black point is the sensor origin o. The blue and red points are the perfectly sampled and distorted points, respectively. The real centroid of the surface $\bar{p}_S$ is displayed in green. (c) and (d) illustrate the error of the voxel centroid of the distorted points $\bar{p}_D$ to the surface as a function of #points n and distance d. (e) and (f) show the Euclidean distance of $\bar{p}_D$ to the surface and the distance of $\bar{p}_D$ to $\bar{p}_S$ as the mean over the distance d.

Figure 3.6 shows the planar surface at a distance $d = 2\,\mathrm{m}$ for the actuator imprecision $\sigma = 0.1$ in (a) and $\sigma = 5.0$ in (b). The results are presented column-wise. As an example and for illustrative purposes, 3000 points $\mathcal{P}$ (blue) were uniformly sampled on the surface. In red color are the corresponding distorted counterparts $\hat{\mathcal{P}}$, i.e., one distorted red point for each sampled blue point. The green point is the real centroid $\bar{p}_S$ of the planar surface and the black point represents the LiDAR sensor at position o and is the center of rotation for the actuator imprecision.

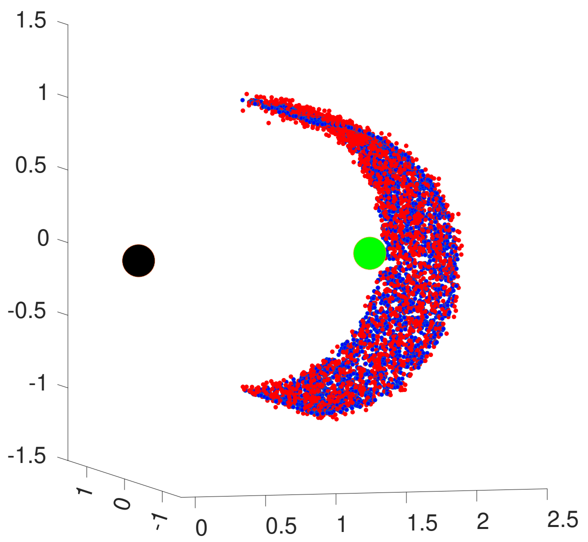
The plots (c) and (d) of Figure 3.6 show the distance of $\bar{p}_D$ to the surface as a function of #points and the distance d. Since random processes are involved, the experiment for each parameter configuration is repeated 1000 times, and the mean is presented. It can be seen that a higher number of sampled points leads to a lower error between the centroid and the surface. For low actuator imprecision, the error is only increasing very slowly with increasing distance d as can be seen in (c). With higher actuator imprecision, the error of the centroid to the surface increases rapidly (d). The plots (e) and (f) illustrate the Euclidean distance of the centroid $\bar{p}_D$ to the surface and the distance of $\bar{p}_D$ to $\bar{p}_S$ as the mean over the distance d. In both plots, the error decreases with an increasing number of points within the voxel. On the other hand, the error is much higher in (f). This can be explained with the increasing error over the distance d in (d). In (e), the distance of $\bar{p}_D$ to the surface is very low, however, when few points are in the voxel, the error is still quite high and is decreasing with the number of points.

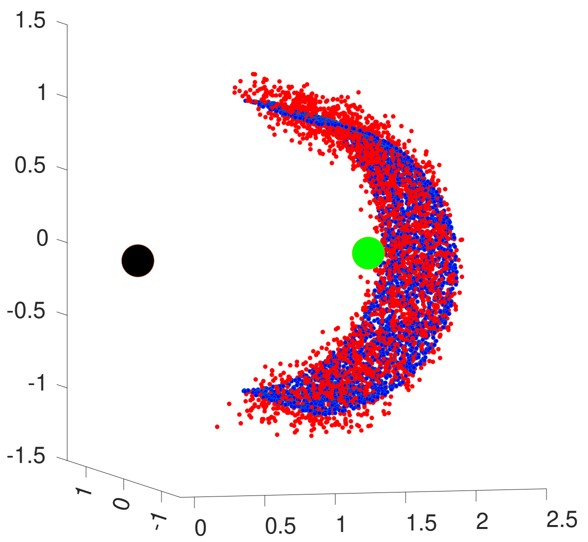
Summarizing the results for a planar surface, it can be said that the actuator imprecision has almost no effect on the distortion of the voxel centroid $\bar{p}_D$ for $\sigma = 0.1$. For a higher imprecision, e.g., $\sigma = 5.0$, the error increases rapidly with the distance of the voxel/surface to the LiDAR sensor. The distance d mainly alters the effect of the actuation distortion on $\bar{p}_D$. A major influence is the number of points on the surface. The error decreases significantly until at least 20 points in the voxel. For $\sigma = 0.1$ and ≥ 10 points inside the voxel, the distance of $\bar{p}_D$ to the surface is smaller than $1\,\mathrm{cm}$ and therefore represents the underlying surface very well, even at higher distances to the LiDAR sensor. However, ≥ 50 points in the voxel are required in the given example for $||\bar{p}_D - \bar{p}_S||_2 < 10\,\mathrm{cm}$.

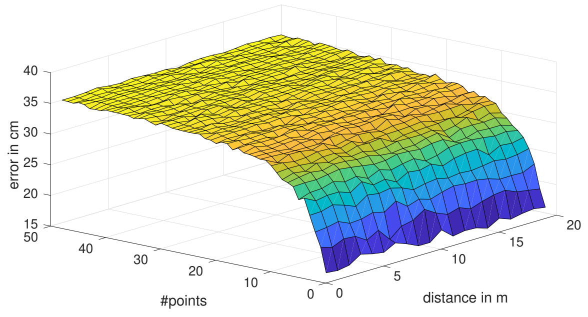
Curved surface

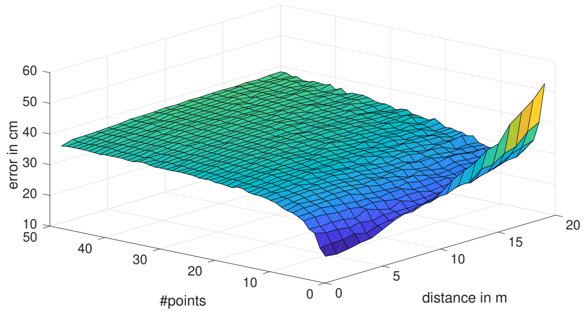
For the evaluation of curvatures, a section of a unit circle is used as surface. Similar to the planar surface, a number n of points $\mathcal{P} = \{p_1, p_2, \cdots, p_n\}$ is uniformly sampled on the curved surface, e.g., similar to a segment of a jet engine of an airplane or the fuselage. The sampled points can take values for $x = [d - r + \cos(\arcsin(v)), d]$, $y = [-v, +v]$ and $z = [-v, +v]$ with $v = 1\,\mathrm{m}$ for a voxel size of $200\,\mathrm{cm}$ and $r = 1$ being the unit radius. The sampled points $\mathcal{P}$ are distorted according to Eq. 3.3 and Eq. 3.4, yielding $\hat{\mathcal{P}}$.

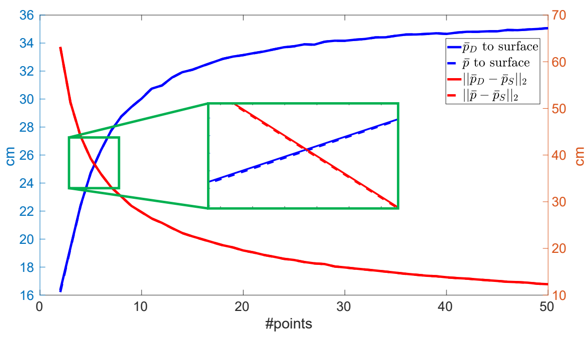
Figure 3.7 shows the curved segment at a distance $d = 2\,\mathrm{m}$ and the results in columnwise order for the actuator imprecisions $\sigma = 0.1$ and $\sigma = 5.0$. (a) and (b) illustrate that even the real surface centroid $\bar{p}_S$ (green) does not lie on the surface. For a surface with less curvature, the centroid converges towards the surface, i.e., as in the example of the planar surface above. (c) and (d) show the distance of the centroid $\bar{p}_D$ to the surface as a function of #points and distance d. The results presented in the figure are the mean values of 1000 repetitions of the random processes with the same parameter configuration. Just like for the


(a) 3000 points with actuator $\sigma = 0.1$

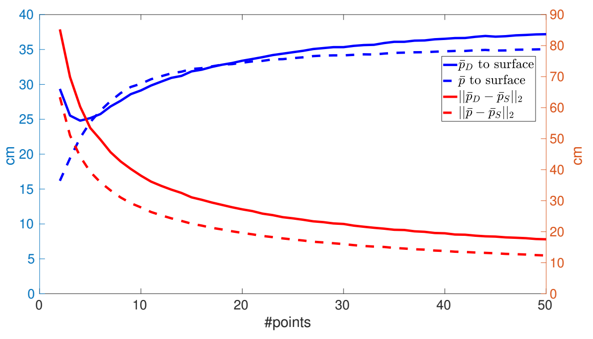

(b) 3000 points with actuator $\sigma = 5.0$


(c) Distance of $\bar{p}_D$ to the surface for actuator $\sigma = 0.1$


(d) Distance of $\bar{p}_D$ to the surface for actuator $\sigma = 5.0$



(e) Distance of $\bar{p}_D$ and $\bar{p}$ to the surface and $\bar{p}_S$ for actuator $\sigma = 0.1$



(f) Distance of $\bar{p}_D$ and $\bar{p}$ to the surface and $\bar{p}_S$ for actuator $\sigma = 5.0$

Figure 3.7: Random uniform generation of point samples on a curved segment with actuator $\sigma = 0.1$ (a) and $\sigma = 5.0$ (b). The black point is the sensor origin o. The blue and red points are the perfectly sampled and distorted points $\mathcal{P}$ and $\hat{\mathcal{P}}$, respectively. The centroid of the blue points $\bar{p}_S$ is displayed in green. (c) and (d) illustrate the error of $\bar{p}_D$ to the surface as a function of the number of sampled points n and distance d. (e) and (f) show the Euclidean distance of $\bar{p}_D$ and $\bar{p}$ to the surface and to $\bar{p}_S$ as the mean over the distance d.

planar surface, an increase in d has no visible effect if the actuator standard deviation is low, e.g., $\sigma = 0.1$ (c). Higher actuation imprecision leads to an increase in error for higher d, if the number of points in the voxel is low as can be seen in (d). (e) and (f) illustrate the distance of $\bar{p}_D$ and $\bar{p}$ to the surface and to $\bar{p}_S$ as a function of #points and the mean over the distance d. For low actuation errors (e), there is almost no difference between $\bar{p}_D$ and $\bar{p}$. Despite an increasing distance to the surface, the centroids converge to the real centroid $\bar{p}_S$ of the surface. However, a centroid lying on the surface itself may be more beneficial for SLAM algorithms, since the distance between LiDAR measurements on the surface and the centroid is minimized during the map optimization process. (f) shows an increased error between $\bar{p}_D$ and $\bar{p}$ due to the higher actuator imprecision. However, both centroids converge to $\bar{p}_S$ when more points are inside the voxel.

Summarizing the results for a curved surface inside a voxel, the results show that neither the centroid of sampled points without distortion $\bar{p}$, nor the real centroid of the surface $\bar{p}_S$ lies on the surface. A higher number of points inside the voxel increases the distance of $\bar{p}_D$ to the surface, while it is converging to the real centroid $\bar{p}_S$ of the surface. Low actuation imprecision with $\sigma = 0.1$ and the distance d have almost no effect on the error of $\bar{p}_D$ to $\bar{p}$. On the other hand, high imprecision increases this error.

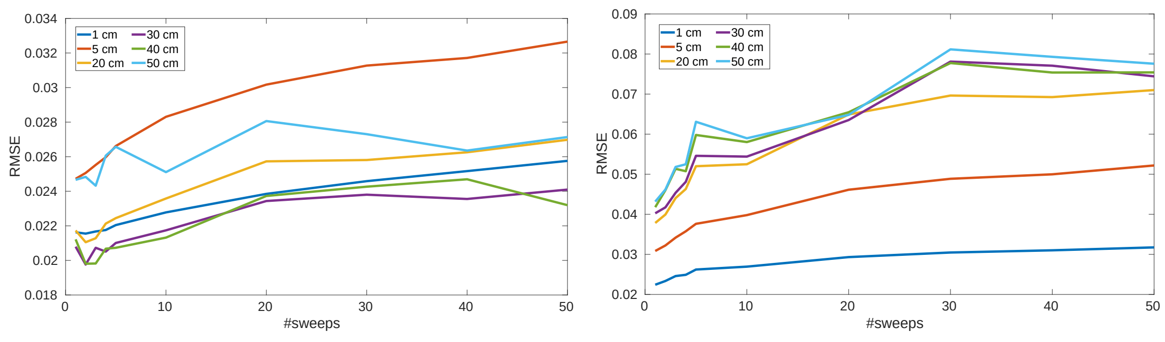
For the planar as well as for the curved surface segment, the centroid of the distorted points $\bar{p}_D$ converged towards $\bar{p}_S$ with increasing #points inside the voxel.

Not a trivial question to answer is, whether the centroid $\bar{p}$, $\bar{p}_S$ or a point directly on the curved surface is a better representation of the surface. Ultimately, this depends on the application.

3.4.1.2 3D triangular mesh

Instead of a single voxel, this section investigates the distance of centroids to a whole mesh surface using the airplane and Tower Bridge models of Dataset 1 (Fig. 3.5(a)) and Dataset 2 (Fig. 3.5(b)) as a function of the number of sweeps (#sweeps). Dataset 1 features the airplane mesh with a lot of curvature on the fuselage, while Dataset 2 contains mainly planar faces on the walls of the towers but also many edges. The setup consists of a virtual gimbal-mounted Velodyne VLP-16 LiDAR sensor as described in Section 3.3. Following up on the findings for a single voxel, the overall RMSE of the centroids to a full mesh surface is dependent on the edge length or size of the voxels and the surface curvature inside of them. Assuming the same mesh and surface for different voxel sizes, the centroids of smaller voxels are expected to be closer to the mesh surface and have a lower RMSE.

An experiment was conducted to compute the RMSE between the voxel centroids and the mesh surface as a function of #sweeps. The LiDAR points were subject to $\sigma = 0.03$ ranging and $\sigma = 0.1$ actuator errors before the centroid computation. The general assumption is that more accumulated points from more sweeps compensate for ranging errors and actuator imprecision. The LiDAR sensor is actuated with an average speed of $9.6\,\mathrm{deg/s}$ between $\pm 0.6\,\mathrm{rad}$ around the roll axis. Hence, one sweep consists of 96 scans on average. Figure 3.8 shows the results of the experiment for different voxel sizes using Dataset 1 (a) and Dataset 2 (b). It can be seen that for both datasets, there is a trend of increasing RMSE with increasing #sweeps, independently of the voxel size. For the sake of the voxel centroids being close to the actual mesh surface, it is, therefore, advisable to choose a low number of sweeps. However, a clear



(a) Dataset 1 - Airplane (b) Dataset 2 - Tower Bridge

Figure 3.8: RMSE of the voxel centroids to the mesh surfaces as a function of the number of sweeps. The legend contains the color code for the voxel edge length. The measured points were subject to $\sigma = 0.03$ ranging errors and $\sigma = 0.1$ actuator error before the centroid computation.

difference can be seen between the two datasets regarding the voxel sizes. Dataset 1 shows the lowest RMSE when choosing voxel sizes of $30\,\mathrm{cm}$ and $40\,\mathrm{cm}$, while $5\,\mathrm{cm}$ shows the highest error. While it is difficult to gain insight into why this is the case, it shows that the best voxel size is dependent on the underlying structure. In comparison, Dataset 2 shows the expected results: A smaller voxel size results in a lower RMSE. Still, the error is increasing with more points in the voxels when capturing more sweeps. Despite Dataset 2 does not feature much curvature, this may be caused by the edges and fine details of the Tower Bridge mesh.

3.4.2 LOAM with a static self-generated initial map

The previous experiments showed the influence of actuator distortion, the number of sweeps, and voxel sizes on the centroid-to-surface distance. However, this distance may not negatively influence the pose estimation by the LOAM algorithm. In this section, the results for different voxel sizes, different numbers of sweeps during initial map creation, and varying actuator errors are presented. Finally, the proposed approach is benchmarked using all three datasets as shown in Fig. 3.5.

In the experimental results, the proposed approach is termed *Init-LOAM-F* (for fixed). It creates an immutable initial map, which can not be changed during the exploration. New points are only added to voxels, which are not occupied by points of the initial map. Karimi et al. [3] presented a method for initial map creation, which leaves the points mutable. In the experiments, this method is termed *Init-LOAM-V* (for variable), where the voxel centroids of the initial map are updated during the exploration.

The results for this chapter are presented when running in *offline* mode. This means, that only one LiDAR scan is sent into the pipeline at a time, e.g., in an on-demand manner. Once the final map-optimized pose from the mapping module is available and the scan points are inserted, a signal is sent to request the next LiDAR scan. This ensures that the pipeline can run at its own speed and will therefore not perform frame dropping. In this mode, all LiDAR scans of the dataset are processed and inserted into the map. The main advantage of this mode is reproducibility since no LiDAR scans are randomly dropped.

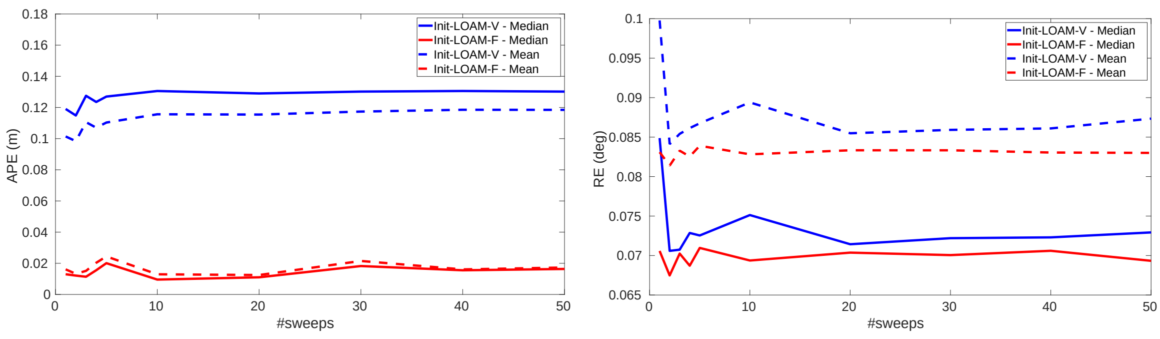
Table 3.2: Results of LOAM for different voxel edge lengths on Dataset 1. The parameter e defines the voxel size for the corner feature map $\mathcal{P}_{\mathcal{E}}$ and h for the surface feature map $\mathcal{P}_{\mathcal{H}}$, respectively. The best results are marked in **bold**.

e (in cm)	h (in cm)	APE (in cm)			RE (in deg)			map time (in ms)
		max	mean	median	max	mean	median	mean
5	10	46	19	19	2.29	0.55	0.55	5010
10	20	33	18	19	2.29	0.55	0.55	2102
20	40	**20**	**14**	**15**	1.99	**0.17**	**0.18**	747
30	60	101	52	47	2.85	2.18	2.19	451
40	80	46	26	23	**1.37**	1.03	1.03	286
50	100	66	34	29	2.19	1.52	1.52	**185**

3.4.2.1 Analysis for different voxel sizes

Before the results for the proposed approach are presented, the effect of different voxel sizes on the APE for the conventional LOAM algorithm is investigated. Specifically, the values for the parameters e and h as described in Section 3.2 are varied for Dataset 1. e and h control the voxel edge length for the corner and surface feature map, respectively. The results in Section 3.4.1.2 have shown, that the RMSE of the voxel centroids to the surface is dependent on the surface structure. In the case of the airplane model of Dataset 1, the voxel sizes of $30\,\mathrm{cm}$ and $40\,\mathrm{cm}$ have shown the lowest RMSE. However, these voxel sizes may not necessarily be optimal for pose estimation and mapping.

As an example, experiments were conducted on Dataset 1 with varying voxel sizes using the conventional LOAM algorithm [12]. As described in the previous sections, LOAM uses two separate maps for corner and surface features. The default parameters of A-LOAM are $e = 20\,\mathrm{cm}$ and $h = 40\,\mathrm{cm}$. Table 3.2 shows the results of the experiments with $h = 2 * e$, for $e = \{5, 10, 20, 30, 40, 50\}$. It can be seen that a voxel size $e < 20\,\mathrm{cm}$ has no improved APE compared to $e = 20\,\mathrm{cm}$, but a significantly increased mapping time. This is due to the higher number of features in the maps which need to be considered for correspondence estimation and also map optimization. The voxelization of map points also compensates for small scan insertion errors due to slightly wrong transform estimates of the map optimization. This may move the voxel centroids slowly to an incorrect position within the voxels, but is not immediately affected by a wrong scan insertion. However, this only holds for small transform errors. For higher errors scan points are not even assigned to the same voxel and instead may even occupy a new voxel triggering the "double wall" effect. In this regard, larger voxel sizes may be more robust. On the other hand, this also decreases the accuracy of the pose estimation, since fewer correspondences are used in the map optimization and the centroids are only approximations within a larger voxel. This can be seen for the results $e > 20\,\mathrm{cm}$. The highest error can be seen for $e = 30\,\mathrm{cm}$, however, no general conclusion can be drawn from this. SLAM is a highly complex process and incorrect estimation of a single pose with a wrong scan insertion can have a negative effect on all the following exploration, especially indoors, when the robot stays in the same environment.



(a) Absolute Pose Error (b) Rotational Error

Figure 3.9: Comparison of the APE (a) and RE (b) for Init-LOAM-V and Init-LOAM-F as a function of #sweeps for the initial map creation. The results are presented for Dataset 1. The scan points are subject to Gaussian ranging errors with $\mathcal{N}(0, 0.03^2)$ and the actuator transforms are subject to $\mathcal{N}(0, 0.1^2)$.

To summarize the results, $e = 20\,\mathrm{cm}$ and $h = 40\,\mathrm{cm}$ shows the best performance on Dataset 1 and also has a reasonable average mapping time with $747\,\mathrm{ms}$, which ensures a map-optimized pose with $> 1\,\mathrm{Hz}$ as in the original LOAM publication [12]. With a LiDAR sensor providing $10\,\mathrm{Hz}$ LiDAR scans, 1 out of 10 scans will be used to obtain a map-optimized pose and will be inserted into the map. Note that all scans in between will still be processed by the odometry module, providing high-frequency scan-to-scan pose estimates with $10\,\mathrm{Hz}$. For a short amount of time, this is sufficient for real-time robot control but suffers from severe drift until a map optimization occurs to correct the most recent pose estimate.

For all following experiments in this thesis, the voxel size parameters are set to $e = 20\,\mathrm{cm}$ and $h = 40\,\mathrm{cm}$.

3.4.2.2 Analysis for different number of sweeps

As shown in the previous sections, the number of scan points within a voxel influence the centroid position and the description of the underlying surface. The RMSE of the voxel centroid to the surface consistently increased in the experiments of Section 3.4.1.2 with increasing sweep number. Hence, the question arises whether an initial map, which was created with more LiDAR sweeps, also leads to a worse performance of the LOAM algorithm and therefore to a higher APE or RE.

Figure 3.9 shows the results for Init-LOAM-V and Init-LOAM-F as a function of #sweeps. For Init-LOAM-V a slight increase in APE (a) can be seen with increasing #sweeps. The proposed approach Init-LOAM-F is rather constant over the whole range with a slight minimum at 10 sweeps. The same can be seen when evaluating the RE (b). For Init-LOAM-V there is a significant drop in RE for > 1 sweep. However, it should be noted that the RE for Init-LOAM-V and Init-LOAM-F is below $0.1\,\mathrm{deg}$ for all #sweeps. In this experiment, Init-LOAM-F (APE $\approx 2\,\mathrm{cm}$) consistently performs better than Init-LOAM-V (APE $\approx 13\,\mathrm{cm}$). It can be concluded that for low actuator and ranging errors, even a low number of sweeps may be sufficient for the initial map creation.

For the following experiments, the LiDAR scans points of three sweeps are collected to create the initial map for Init-LOAM-V and Init-LOAM-F.

3.4.2.3 Analysis for different actuator errors

Actuator measurement errors have a negative impact on the initial map creation. The proposed approach relies on the high quality of the initial map, since it is created by accumulating LiDAR scans without optimization, i.e., scans are directly inserted into the initial map with the actuator transforms. Hence, the optimization process of LOAM might compensate for higher actuator errors and result in a better performance than the proposed approach with a static initial map, created with inaccurate actuator readings.

In the following experiments, the transform of the actuator reading (Alg. 1, line 6) is augmented with a random Gaussian noise H as in Eq. 3.3. The rotation matrix can then be computed with the random angle α drawn from the distribution. The augmented transform ${}^{L_t}\boldsymbol{T}_{\hat{L}_t}$ is assembled with the rotation matrix and a zero-translation vector. The augmented actuator transform for a LiDAR scan at time t is then described as:

$$ {}^{D_t}\boldsymbol{T}_{\hat{L}_t} = {}^{D_t}\boldsymbol{T}_{L_t}{}^{L_t}\boldsymbol{T}_{\hat{L}_t} \tag{3.7} $$

${}^{D_t}\boldsymbol{T}_{L_t}$ is the perfect actuator transform obtained from the simulation. This augmentation is performed for all actuator readings, i.e., during the initial map creation (Phase 1) as well as during the exploration (Phase 2).

Table 3.3 shows the results of LOAM, Init-LOAM-V and Init-LOAM-F on Dataset 1 for different actuator errors. For Init-LOAM, the initial maps are created by the accumulation of three sweeps. Init-LOAM-F has a very low APE ($\approx 1\,\mathrm{cm}$) for $\sigma = 0.1$ and still performs better than LOAM and Init-LOAM-V until the actuator error reaches $\sigma = 3.0$. For $\sigma = 5.0$, Init-LOAM-V performs best, while Init-LOAM-F has a higher APE than LOAM. This is very interesting since the major difference to Init-LOAM-F is that the voxels created during the initialization are modified and updated during the exploration, i.e., all scan points are inserted into the dynamic map and merged with the voxel centroids. Hence, the actuator-distorted voxels acquired during initialization are effectively corrected by the map optimization process for Init-LOAM-V. At the same time Init-LOAM-V performs better than pure LOAM with map optimization without initialization (Phase 1). It should be noted that the results presented for LOAM also leverage the known actuator transform in the odometry module as described in Alg. 2. The results for $\sigma = 10.0$ show that creating an initial map is not beneficial anymore and even harmful. Init-LOAM-V and Init-LOAM-F perform worse than standard LOAM with respect to APE and RE.

To summarize the results it can be said that the proposed approach Init-LOAM-F is robust to actuator errors up to $\sigma = 3.0$. Considering that current Dynamixel actuators have an imprecision of $0.1\,\mathrm{deg}$, Init-LOAM-F can be employed on real-world robots with standard actuators. The advantages of an initial map vanish for $\sigma > 5.0$ and for $\sigma \geq 10.0$ it is not advisable to create an initial map.

For the following experiments an actuator error of $\sigma = 0.1$ is applied.

Table 3.3: Experimental results for actuator augmentation using Dataset 1 with $\sigma = \{0.1, 0.5, 1.0, 3.0, 5.0, 10.0\}$ in deg and 3 sweeps for Phase 1. The best results are marked in **bold**. Partially reproduced and modified from [5], ©2021 IEEE.

act.	**method**	**APE in** cm			**RE in** deg		
σ		max	mean	median	max	mean	median
0.1	LOAM [12]	26	16	16	2.15	0.36	0.36
	Init-LOAM-V	16	11	13	**0.39**	0.09	**0.07**
	Init-LOAM-F	**8**	**2**	**1**	0.41	**0.08**	**0.07**
0.5	LOAM [12]	21	15	15	**2.10**	0.47	0.40
	Init-LOAM-V	19	14	15	2.14	0.41	**0.34**
	Init-LOAM-F	**14**	**3**	**2**	2.13	**0.40**	**0.34**
1.0	LOAM [12]	**24**	18	20	4.70	0.84	0.71
	Init-LOAM-V	30	24	25	**4.02**	**0.81**	**0.67**
	Init-LOAM-F	25	**9**	**8**	4.33	0.82	0.68
3.0	LOAM [12]	78	43	41	13.19	2.76	2.36
	Init-LOAM-V	44	32	32	12.58	2.47	2.08
	Init-LOAM-F	**26**	**10**	**10**	**12.29**	2.44	**2.05**
5.0	LOAM [12]	238	127	117	23.88	5.85	5.27
	Init-LOAM-V	**167**	**87**	**76**	**21.94**	**5.03**	**4.38**
	Init-LOAM-F	275	145	136	23.46	6.40	5.89
10.0	LOAM [12]	**373**	**197**	**180**	46.82	**10.50**	**9.18**
	Init-LOAM-V	580	309	287	**46.78**	13.17	12.28
	Init-LOAM-F	801	421	390	58.38	17.14	16.64

3.4.2.4 Benchmark using all datasets

This section presents the results for LOAM, Init-LOAM-V, and the proposed Init-LOAM-F on all three datasets. On one hand, Init-LOAM is compared to LOAM operating in frame $\{D\}$, which leverages the actuator transform in the odometry estimation module as described in Alg. 2. Here, the map optimization is performed in the robot frame $\{D\}$. On the other hand, Init-LOAM is benchmarked against the original LOAM algorithm operating in frame $\{L\}$. It processes the scans in the LiDAR frame $\{L\}$, without leveraging the knowledge about the actuator transform. Hence, the LiDAR rotation is fully estimated by the odometry and map optimization.

Table 3.4 shows the results on all three datasets. For Dataset 1 all methods perform well with an APE $< 30\,\mathrm{cm}$. LOAM with scans processed in the LiDAR frame $\{L\}$ is not subject to actuator reading errors, since no additional transform is performed. However, for the methods leveraging the actuator transform (operating in frame $\{D\}$), a zero-mean Gaussian noise of $\sigma = 0.1$ is applied. Creating an immutable initial map for Init-LOAM-F performs exceptionally well with a median APE of only $1\,\mathrm{cm}$. The same can be seen for Dataset 2, where an initial map prevents heavy drift compared to LOAM. Init-LOAM-F also performs

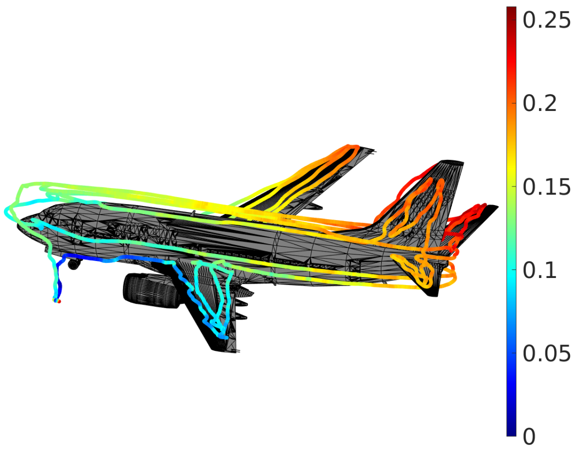
Table 3.4: Experimental results for LOAM, Init-LOAM-V, and Init-LOAM-F on all three datasets. The best results are marked in **bold**. Adapted and modified from [5], ©2021 IEEE.

	method	frame	act. σ	#sweeps	APE in cm			RE in deg		
				Phase 1	max	mean	median	max	mean	median
Dataset 1	LOAM [12]	$\{L\}$	-	-	29	16	17	**0.10**	**0.02**	**0.02**
	LOAM [12]	$\{D\}$	0.1	-	26	16	16	2.15	0.36	0.36
	Init-LOAM-V	$\{D\}$	0.1	3	16	11	13	0.39	0.09	0.07
	Init-LOAM-F	$\{D\}$	0.1	3	**8**	**2**	**1**	0.41	0.08	0.07
Dataset 2	LOAM [12]	$\{L\}$	-	-	740	549	609	13.41	11.87	11.99
	LOAM [12]	$\{D\}$	0.1	-	523	363	381	12.74	7.88	7.91
	Init-LOAM-V	$\{D\}$	0.1	3	69	10	8	0.73	0.23	0.23
	Init-LOAM-F	$\{D\}$	0.1	3	**68**	**7**	**6**	**0.64**	**0.20**	**0.19**
Dataset 3	LOAM [12]	$\{L\}$	-	-	1512	495	499	10.09	3.66	2.47
	LOAM [12]	$\{D\}$	0.1	-	1539	457	439	10.24	3.30	2.27
	Init-LOAM-V	$\{D\}$	0.1	3	1387	389	351	**9.26**	2.86	**2.11**
	Init-LOAM-F	$\{D\}$	0.1	3	**1267**	**292**	**238**	9.94	**2.68**	2.85

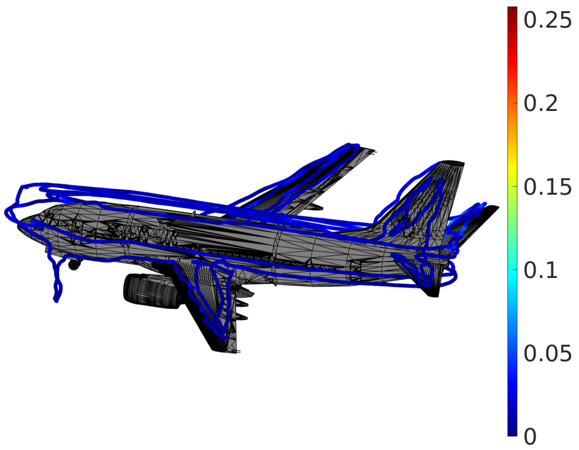
best in Dataset 3, despite the challenging outdoor environment at the Eiffel Tower. Here, all methods have the highest APE at over $10\,\mathrm{m}$, which is large considering that Dataset 3 has the shortest flight trajectory of all datasets (see Table 3.1). Summarizing the results, creating an initial map proved beneficial for all three datasets. Init-LOAM-F with the immutable initial map shows the best performance compared to Init-LOAM-V and conventional LOAM. Leveraging the actuator transform reduces the APE in all cases compared to LOAM processing the scans in the LiDAR frame $\{L\}$.

Figure 3.10 visualizes the results for Dataset 1. (a) and (b) show the APE mapped onto the trajectory of Dataset 1 for LOAM operating in $\{D\}$ and Init-LOAM-F, respectively. A continuous drift can be seen for LOAM towards the rear of the airplane, while Init-LOAM-F shows a very low APE around the whole airplane. (c) and (d) show the final surface feature maps and the airplane model as triangular mesh. For LOAM, all surface features are inserted into a single dynamic map. In fact, all points shown are the voxel centroids of the surface map with a voxel edge length of $h = 40\,\mathrm{cm}$, as discussed above. In (c), all centroids are continuously updated after each scan insertion. In contrast, (d) shows the initial (blue) and dynamic (red) surface feature maps. The blue points were processed by Phase 1 of the proposed pipeline and the red points by Phase 2. The former were acquired from the starting position of the UAV, while it was still stationary and the latter were acquired and inserted during the exploration. Hence, the red points were not visible from the starting position. A "V"-shape can be seen on the right side of the hangar between the initial and dynamic map points. This is due to the actuation limit of $\pm 0.6\,\mathrm{rad}$ of the LiDAR sensor during initial map creation. In (d), above the airplane on the back wall of the hangar, a clear separation with a fine white space is visible between the points of the initial and dynamic map. The proposed scan insertion into the dynamic map as illustrated in Fig. 3.3 leads to this effect.

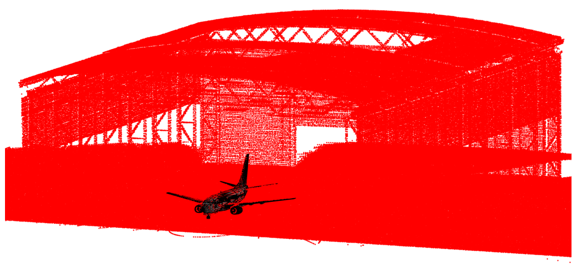
Figure 3.11 illustrates the APE mapped onto the trajectory and final surface feature maps of Dataset 2 using LOAM operating in $\{D\}$ (a), and Init-LOAM-F (b). Without the initial map, wrong scan insertions are more likely and may lead to a skewed map representation



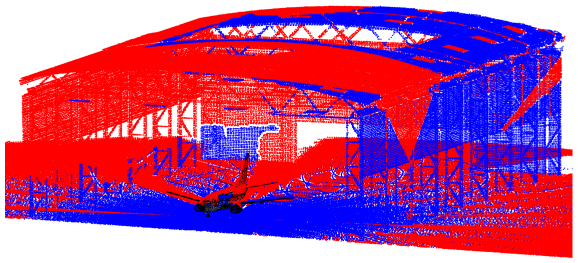
(a) APE (m) using LOAM in $\{D\}$



(b) APE (m) using Init-LOAM-F

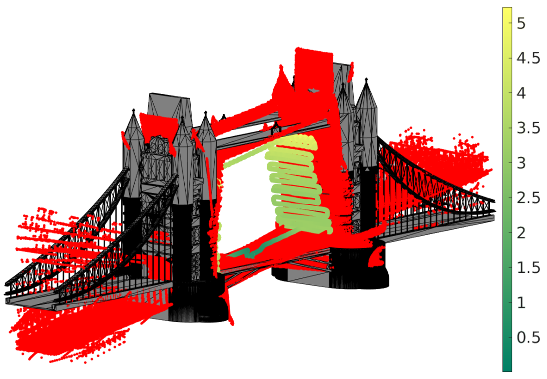


(c) Final surface feature map using LOAM without initial map (a)

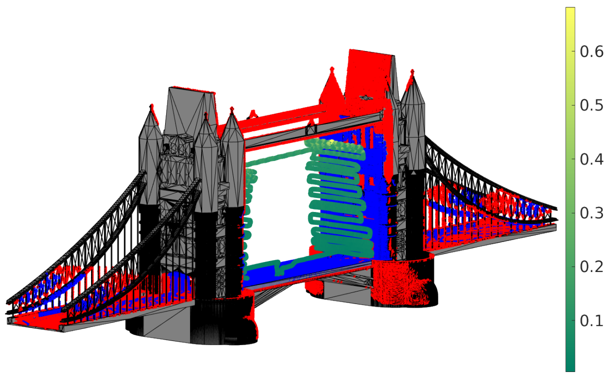


(d) Immutable initial (blue) and dynamic (red) surface feature maps using Init-LOAM-F (b)

Figure 3.10: Visualization of the APE mapped onto the trajectory of Dataset 1 for LOAM operating in $\{D\}$ (a), and Init-LOAM-F (b). (c) and (d) show the final surface feature maps, including the airplane model as triangular mesh. The front of the hangar was removed for illustrative purposes.



(a) APE (m) using LOAM in $\{D\}$ and dynamic surface feature map



(b) APE (m) using Init-LOAM-F and initial (blue) and dynamic (red) surface feature maps. Reproduced from [5], ©2021 IEEE.

Figure 3.11: Visualization of the APE mapped onto the trajectory of Dataset 2 for LOAM operating in $\{D\}$ (a), and Init-LOAM-F (b). The final surface feature maps were overlaid with the triangular mesh of the Tower Bridge. The ground was removed for illustrative purposes.

and strong drift, as can be seen in (a). The maximum error of over $5\,\mathrm{m}$ is at the highest point of the trajectory due to accumulated drift at the right tower. In contrast, the initial map (blue) in (b) prevents the wrong scan insertion and effectively guides the localization and mapping process during exploration. Here, the highest error is also at the top of the trajectory with $\approx 68\,\mathrm{cm}$ where few parts of the initial map are visible and the localization mainly needs to rely on points inserted during exploration (Phase 2).

3.5 Chapter summary

This chapter presented a modification of the conventional LOAM framework to enable the generation of an immutable initial map. The core idea of the proposed approach is that an initial map can be created with much higher accuracy from the starting position of the robot by accumulating LiDAR scans with the known actuator transforms. This assumes that the sensor is mounted on a gimbal/actuator. The proposed pipeline is divided into two phases: initialization (Phase 1) and exploration (Phase 2). This chapter gave a detailed description of the modifications introduced to the LOAM algorithm to make the initial map immutable during the exploration while being able to continue mapping unseen parts of the environment.

The experimental setup included a simulated UAV as a robotic platform, following predefined trajectories for visual inspection. A virtual Velodyne VLP-16 LiDAR sensor was mounted on the UAV via a gimbal, capturing LiDAR scans during the initialization and exploration phases. The LiDAR was continuously actuated back and forth.

First, the effects of actuator reading imprecision and ranging errors were evaluated on a planar and curved surface within a single voxel. The results for a planar surface have shown, that an increased number of points within the voxel decreases the distance between the centroid of distorted points and the underlying surface. Also, the distance to the real surface centroid decreases. However, in the case of a curved surface, even the centroid of undistorted points does not lie on the surface. A voxel centroid of a higher number of distorted points converges towards the real surface centroid, but the distance to the surface increases. The influence of #sweeps and different voxel sizes was then evaluated by means of RMSE on a triangular mesh. The optimal voxel size varied for Dataset 1 and Dataset 2. However, in both cases, the RMSE increased when accumulating more LiDAR scans.

Experiments have demonstrated that the default voxel size parameters of LOAM are optimal for Dataset 1 by means of APE and that accumulating more LiDAR sweeps does not improve localization and mapping performance. Init-LOAM-F has proven robust to actuator errors up to $3.0\,\mathrm{deg}$ in the benchmark against Init-LOAM-V and LOAM. For $> 10.0\,\mathrm{deg}$ it is rather harmful to create an initial map, and generally, the use of the actuator transform is questionable with such high imprecision. In the benchmark, Init-LOAM-V performed better than LOAM in all three datasets, demonstrating that creating an initial before the exploration improves localization and mapping, even if the initial map is changed during the exploration. Ultimately, Init-LOAM-F outperformed Init-LOAM-V and LOAM. Hence, keeping the initial map immutable during the exploration has proven beneficial and led to a reduced

APE.

Chapter 4

Improving LOAM with mesh features of a known 3D reference object

This chapter introduces an approach to improving LOAM [12] accuracy by leveraging prior knowledge about a 3D reference object. With simulated data, the proposed approach is evaluated in three visual inspection scenarios.

Parts of this chapter have been published in [1].

4.1 Problem statement

Instead of an a priori known map before the exploration or the generation of an initial map at the starting position as in Chapter 3, the knowledge about only parts of the environment can be leveraged to improve localization and mapping performance. Often, robotic platforms are deployed in scenarios, where the area of operation is difficult to reach for human workers. For example, windmills, bridges, buildings, or also airplanes need to be inspected regularly for defects. For most of these objects, CAD models exist from the manufacturer or can be created with highly accurate measuring devices. The CAD models can then be used as prior knowledge by the robot. Due to the repetitive nature of visual inspections, the generated 3D models can be reused. Windmills are not fully rigid due to the rotation of the blades but can be put into a maintenance position so as to be consistent with the CAD model. In disaster scenarios, robots are deployed for manipulation or search and rescue missions. Here, architectural or emergency floor plans [93]–[95], [97], [98] can be leveraged as prior knowledge about the interior of a building and used by the robot for global relocalization and improved LOAM accuracy. However, 2D floor plans can mainly only be used in 2D LiDAR-SLAM applications due to the missing height information.

Instead of a single object, a whole 3D CAD building model can also be used as prior knowledge by the robot. No matter if a whole building or a single 3D reference object is used, in order to leverage it as prior knowledge the initial relative position to the reference object in the global coordinate frame of the robot needs to be determined. This assumes that the position of the reference object is not changing, i.e., an airplane or vehicle used as a reference object is not moved during the robot exploration. Since the map of the SLAM algorithm is typically created at the starting position of the robot, the relative pose to the reference

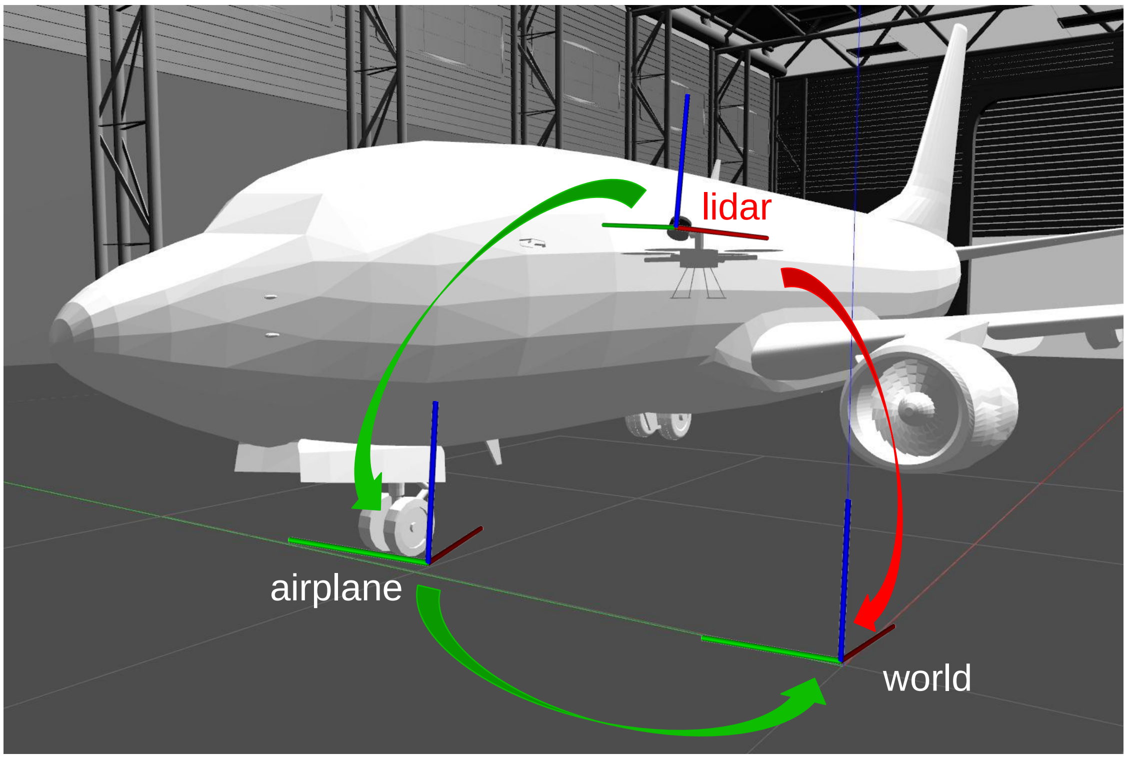


Figure 4.1: Visualization of the concept for relative localization to a 3D reference object using a scene from a simulation. The UAV is equipped with a LiDAR sensor and uses an airplane as a 3D reference object during the exploration. Typically, the transformation *world* → *lidar* is estimated by the LiDAR-SLAM algorithm. The relative transform *airplane* → *lidar* can support this estimation if the position of the airplane in the world frame (*world* → *airplane*) is known.

object needs to be determined from there before the start of the exploration. The starting position also defines the map/world coordinate frame of the LOAM algorithm. If the relative position is fully unknown, global relocalization techniques need to be employed, eventually followed by local registration. With a good initial guess, local registration methods can be used directly to determine the relative pose to the reference object, e.g., using ICP. Hence, the robot should be initially placed close to the reference object in a way that global/local registration techniques can accurately determine the relative transform. For example, if a building model is used for a facade inspection, the robot should be placed in a section with a unique layout to avoid ambiguities. If an airplane or bridge is used, the robot should be placed in an initial position, so that it can scan unique parts of the reference object with the LiDAR sensor.

So far, only a few works have been presented to perform highly accurate ego pose estimation leveraging a 3D reference object as a 3D CAD model as discussed in Section 2.2.2. Sandy et al. [109] leveraged a CAD model for the highly precise positioning of an end-effector for brick stapling. Gawel et al. [110] performed ray tracing into a CAD model for highly accurate drilling operations. However, all these works do not improve mapping quality nor directly integrate the 3D model into the mapping process of the LiDAR-SLAM pipeline.

In this chapter, a method is presented as an extension to the LOAM framework [12] to improve SLAM performance by leveraging knowledge about a known 3D reference object. The proposed method is termed *Reference-LOAM* (R-LOAM). In contrast to previous work, the proposed approach tightly couples the reference object into the SLAM pipeline. The experimental evaluation employs a known CAD model in the form of a triangular mesh as a 3D reference object. Also, the exact position of the reference object in the coordinate frame of the robot is assumed to be known. Figure 4.1 shows the concept of a relative localization between a LiDAR mounted on a UAV and an airplane as the reference object. A SLAM algorithm estimates the transform or current pose of the LiDAR in the world or map frame, depicted as *world* $\rightarrow$ *lidar*. To further support this pose estimate, a relative localization to the airplane is introduced (*airplane* $\rightarrow$ *lidar*). However, the position of the airplane in the world frame (*world* $\rightarrow$ *airplane*) is required to be known with high precision.

The reference object could also be used as a partial initial map as described in Chapter 3. In the case of LOAM [12], however, this is not possible since the maps are actually consisting of corner and surface features (see Section 2.2.1). These are extracted in the scan registration module directly from the LiDAR scan (see Section 2.1.4.1). Hence, a whole point cloud or even triangular mesh can not be converted to feature maps when using the LOAM framework. Besides, SLAM algorithms usually operate on sparse maps to maintain real-time capability. The proposed approach combines the conventional map optimization on sparse maps using point features, with additional mesh features extracted from a triangular mesh of a 3D reference object. Alternatively, the mesh can be replaced by a dense point cloud of the reference object. A triangular mesh can be seen as an infinitely dense point cloud due to its implicit definition of the object surface with faces. Converting the mesh to a point cloud will lose this implicit surface definition.

4.2 Methodology

This section describes the modules added to the conventional LOAM pipeline in detail to enable the usage of point-to-mesh correspondences in addition to conventional point-to-point correspondences in the map optimization step. Figure 4.2 gives a schematic overview of the proposed pipeline. The blue boxes are part of the original LOAM framework as described in Section 2.2.1. The green boxes are the modules of the proposed extension. All the modifications were introduced in the mapping module of the A-LOAM framework. A key property of the extension is that the point-to-mesh correspondence estimation can be performed in parallel to the estimation of conventional point-to-point correspondences, i.e., in a multi-threaded process until it is merged in the joint optimization module. This adds computational overhead but limits the added pose estimation delay to a minimum.

In the following, the modules of the extension are described in detail.

4.2.1 BVH initialization

For the proposed approach, an AABB tree structure is used as BVH. An AABB tree is a space division technique to speed up collision or closest points queries. Here, it is employed to

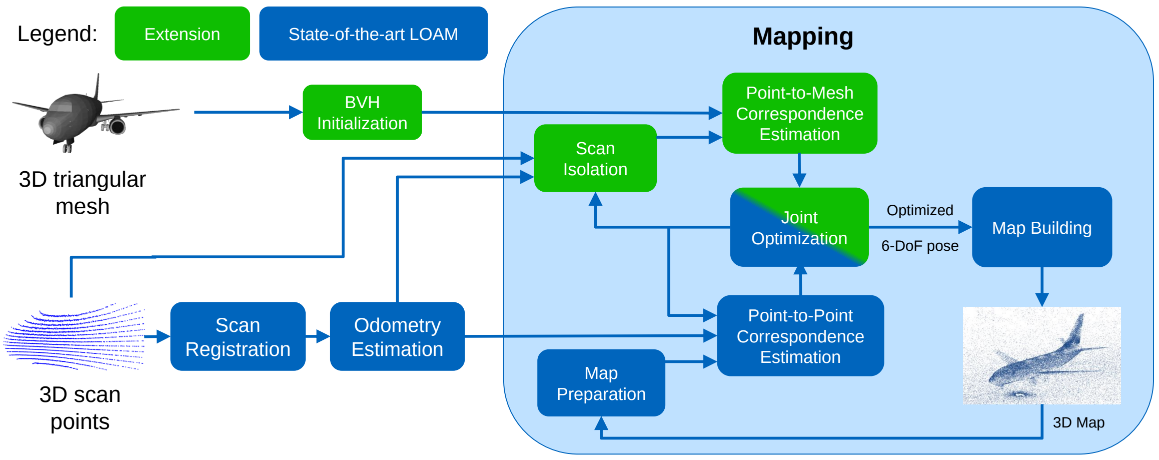


Figure 4.2: Overview of the proposed pipeline to support mesh features in the mapping module. Blue boxes represent unmodified components and green boxes were added to the pipeline. The joint optimization module is the core of the proposed method, where point-to-mesh correspondences are jointly optimized with conventional point-to-point correspondences. Reproduced from [1], ©2021 IEEE.

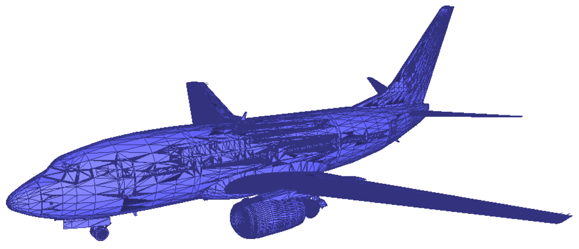
speed up the closest point search between a query point cloud and a triangular mesh. Section 2.1.4.3 described in detail how the closest point on a face of the triangular mesh is found for a query point. If no BVH method is used, all triangles of the mesh have to be tested to find the closest point, which is computationally intractable for real-time applications. Other space division techniques may be used as well, but AABB trees are very common among computer graphics applications.

Initializing the BVH essentially creates the AABB tree structure. For this, the triangular mesh needs to be transformed to the known location in the global frame, e.g., the map/world frame $\{W\}$ of the SLAM algorithm. The location of the mesh as a 3D reference object needs to be determined before the exploration. As Fig. 4.1 has shown, the relative localization *airplane* $\rightarrow$ *lidar* is only beneficial, if *world* $\rightarrow$ *airplane* is exactly known. The vertices of a mesh are defined with respect to its local coordinate frame (*airplane* in the figure). Once the BVH is initialized, no changes to the position of the reference object in the global frame can be made. In the case of an AABB tree, it would need to be reinitialized otherwise. Depending on the complexity (#faces) of the triangular mesh, this can take several seconds.

After the initialization, it can be reused for arbitrary query point cloud to mesh queries. Figure 4.3 illustrates the triangular mesh of the airplane as the reference object (a) and the initialized AABB tree (b).

4.2.2 Scan isolation

The task of the scan isolation module is to remove all points from an incoming LiDAR scan, which do not belong to the surface of the reference object. The module processes the raw LiDAR scan points instead of the extracted features from the scan registration module of LOAM. The LiDAR scan points $\mathcal{P}^{L_t}$ are first transformed to the map frame using the map-integrated odometry estimate ${}^{W}\boldsymbol{T}_{L_t} \leftarrow {}^{W_{\text{odom}}}\boldsymbol{T}_{L_t}$, obtaining $\mathcal{P}^{W_t}$. Now, both the 3D reference


(a) Triangular mesh of an airplane as 3D reference object

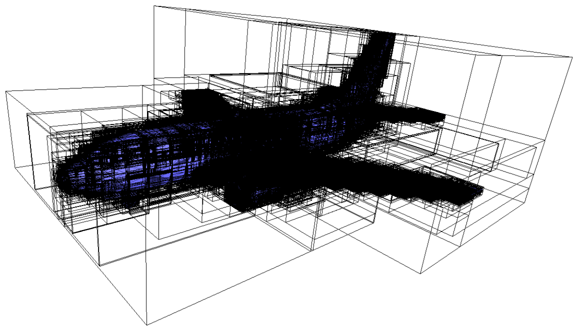

(b) Hierarchical AABB tree structure of the airplane in (a)

Figure 4.3: Illustration of the BVH initialization of an airplane as the reference object (a). The AABB tree (b) is initialized before the exploration after the triangular mesh has been transformed to its location in the global frame.

object and the recent LiDAR scan are in the same global coordinate frame.

To isolate the LiDAR scan points belonging to the reference object from the other points, a cropping method can be employed. In this case, a bounding box cropping method is selected. The bounding box is placed at the known location of the reference object in the global frame and since the current LiDAR scan points $\mathcal{P}^{W_t}$ are in the same frame, they can be easily filtered with a cropping method. To compensate for localization errors of the transform ${}^{W}\boldsymbol{T}_{L_t}$, the bounding box is extended with an additional buffer of $+2\,\mathrm{m}$ at the sides and top of the bounding box. All points below $1\,\mathrm{m}$ height are removed, effectively removing the ground in the scan. The size of the crop box is a trade-off. Alternatively, tighter and more complex cropping methods can be used, e.g., convex hull cropping. However, if the localization errors in ${}^{W}\boldsymbol{T}_{L_t}$ are higher, scan points of the reference object will be removed. On the other hand, leaving too many non-reference object points might decrease the performance in the following steps. Figure 4.4 illustrates the scan isolation process. After the transformation to the map frame, the LiDAR scan is cropped using the extended bounding box. Ideally, all remaining scan points belong to the reference object.

4.2.3 Point-to-mesh correspondence estimation

For each isolated scan point, the closest point on the triangular mesh surface needs to be determined. The initialized AABB tree can be used to efficiently determine the closest face of the mesh for each scan point. For this, the AABB tree is traversed downward to the leaf node containing only a single face by comparing the query point to the extreme points of the AABBs. The closest point on the face can be computed using 3D geometry as described in detail in Section 2.1.4.3. In the following, a closest point on the mesh surface is referred to as a *virtual* point $p_{\mathcal{V}}^{W_t}$. Correspondences can then be created between the isolated scan point cloud $\mathcal{P}^{W_t} = \{p_1^{W_t}, p_2^{W_t}, \cdots, p_n^{W_t}\}$ captured at time t and the virtual point cloud $\mathcal{P}_{\mathcal{V}}^{W_t} = \{p_{\mathcal{V}_1}^{W_t}, p_{\mathcal{V}_2}^{W_t}, \cdots, p_{\mathcal{V}_n}^{W_t}\}$. The set of point-to-mesh correspondences is denoted as $C_{\mathcal{M}} = \{c_{\mathcal{M}_1}, c_{\mathcal{M}_2}, \cdots, c_{\mathcal{M}_n}\}$, with $c_{\mathcal{M}_i} = \langle p_i^W, p_{\mathcal{V}_i}^W \rangle$.

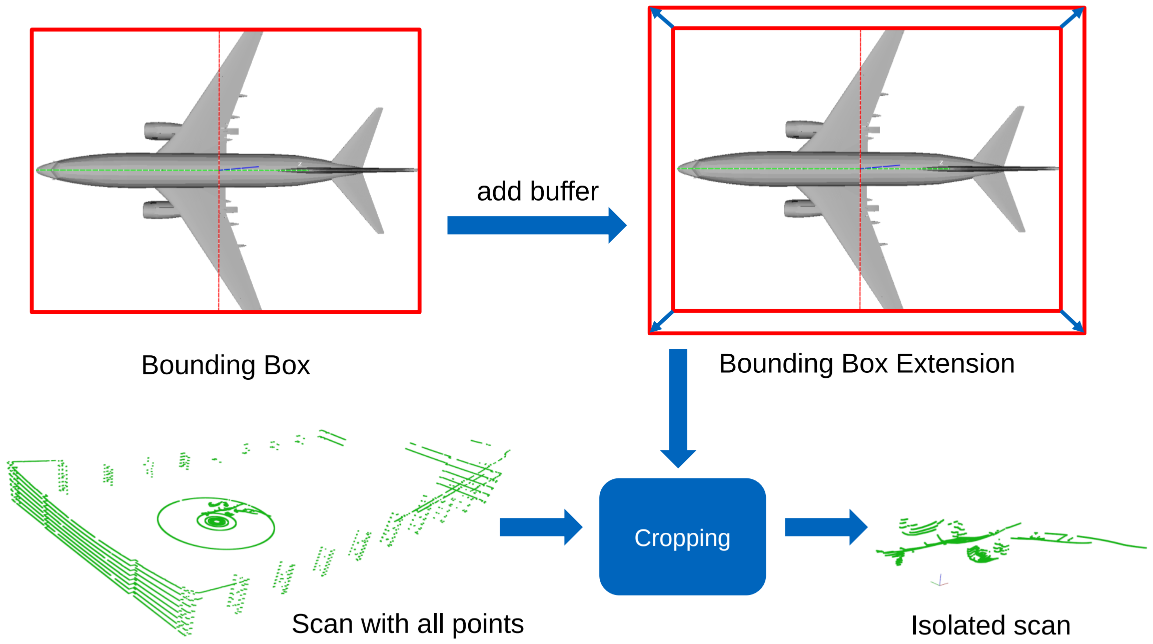
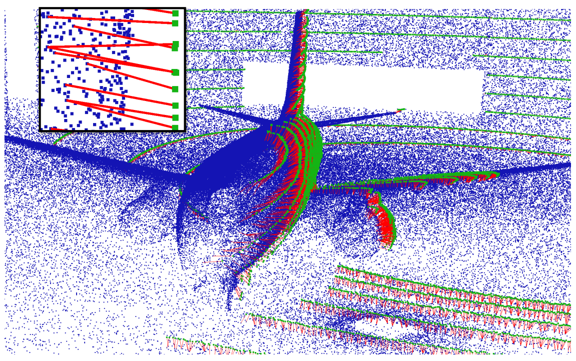
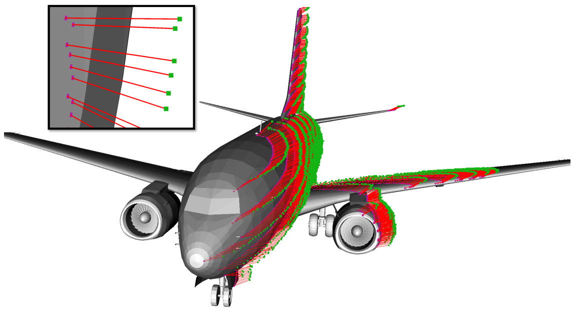


Figure 4.4: Pipeline of the scan isolation process using a bounding box cropping method



(a) Conventional point-to-point correspondences between the current scan (green) and the map points (blue)



(b) Point-to-mesh correspondences between the isolated current scan points (green) and the triangular mesh of the reference object

Figure 4.5: Illustration of the conventional point-to-point correspondences of LOAM (a) and the proposed additional point-to-mesh correspondences between the isolated scan points (green) and the virtual points (purple) on the surface of the triangular mesh as 3D reference object (b). In both figures, the correspondences are visualized as red lines. Adapted from [1], ©2021 IEEE.

The point-to-mesh correspondence estimation is a process running in parallel to the conventional point-to-point correspondence estimation as shown in Fig. 4.2. The latter creates corner and surface features correspondences $C_{\mathcal{E}}$ and $C_{\mathcal{H}}$ (see Section 2.2.1), respectively.

Figure 4.5 illustrates the conventional (a) and proposed point-to-mesh correspondences (b). The red lines depict the correspondence distance. Assuming an exact known location of the reference model in the map frame, this distance is exactly the error of the current pose estimation ${}^{W}\boldsymbol{T}_{L_t}$ with which the current scan points were transformed to the map frame. If the transform is corrected, the scan points will perfectly align to the surface of the reference model, and the pose error further decreases.

The current pose estimate ${}^{W}\boldsymbol{T}_{L_t}$ might be off by several meters in translation or the rotation around roll, pitch, or yaw might be off by a few degrees. In this case, scan points not belonging to the reference object will pass the scan isolation step and will be used to create point-to-mesh correspondences. A simple maximum correspondence distance threshold of $1\,\mathrm{m}$ is applied to overcome this issue. All correspondences with a greater distance are removed. Note that this might ignore all scan points on the reference object, if the translational error is too high, e.g., $> 2\,\mathrm{m}$. However, in practice this is often prevented: Point-to-mesh correspondences are used for every LiDAR scan and optimization step. Before the error gets too high, most point-to-mesh correspondences fall within the maximum correspondence threshold and even with a rotational error, there are usually always some valid point-to-mesh correspondences. Therefore, an unrecoverable drift in the pose estimate ${}^{W}\boldsymbol{T}_{L_t}$ is rather unlikely.

4.2.4 Joint optimization

The conventional point-to-point and proposed point-to-mesh correspondences are added to a joint optimization problem. The state-of-the-art LOAM map optimization was described in Eq. 2.48.

The proposed joint optimization with the total cost J is formulated as

$$\begin{aligned} J(\boldsymbol{q},\boldsymbol{t}) = & \frac{1}{\bar{\bar{C}}_{\mathcal{H}}} \sum_{c_{\mathcal{H}} \in C^{\mathcal{H}}} \rho(\|f_{\mathcal{H}}(c_{\mathcal{H}},\boldsymbol{q},\boldsymbol{t})\|_2^2) \\ & + \frac{1}{\bar{\bar{C}}_{\mathcal{E}}} \sum_{c_{\mathcal{E}} \in C_{\mathcal{E}}} \rho(\|f_{\mathcal{E}}(c_{\mathcal{E}},\boldsymbol{q},\boldsymbol{t})\|_2^2) \\ & + \frac{\lambda}{\bar{\bar{C}}_{\mathcal{M}}} \sum_{c_{\mathcal{M}} \in C_{\mathcal{M}}} \rho(\|f_{\mathcal{M}}(c_{\mathcal{M}},\boldsymbol{q},\boldsymbol{t})\|_2^2)\,. \end{aligned} \tag{4.1}$$

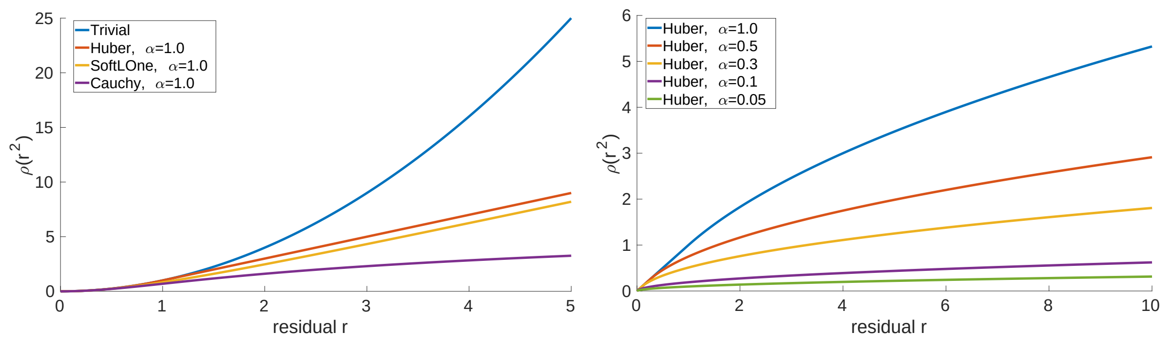
$\boldsymbol{q}$ and $\boldsymbol{t}$ are the quaternion and translational vectors to be optimized. $f_{\mathcal{E}}$ and $f_{\mathcal{H}}$ are the cost functions for the corner and surface features, respectively. They compute the residuals as described in Eq. 2.45 and 2.52. The equation is solved with a Levenberg-Marquardt trust-region algorithm [77], [78]. $f_{\mathcal{M}}$ is the proposed cost function for point-to-mesh correspondences, which is defined as

$$f_{\mathcal{M}}(c_{\mathcal{M}},\boldsymbol{q},\boldsymbol{t}) = r = (\boldsymbol{q}p^{L}\boldsymbol{q}^{-1} + \boldsymbol{t}) - p_{\mathcal{V}}^{W}\ \text{, with } p^{L}, p_{\mathcal{V}}^{W} \in c_{\mathcal{M}}\,. \tag{4.2}$$

Note that here $c_{\mathcal{M}} = \langle p^{L}, p_{\mathcal{V}}^{W} \rangle$. The isolated scan point p has been transformed back to the LiDAR frame $\{L\}$ after the isolation since the optimization process is refining the transform ${}^{W}\boldsymbol{T}_{L}$ by adjusting $\boldsymbol{q}$ and $\boldsymbol{t}$.

Normalization terms were added to Eq. 4.1 when comparing it to the original formulation (Eq. 2.48). $\bar{\bar{C}}_{\mathcal{H}}$ and $\bar{\bar{C}}_{\mathcal{E}}$ are the number of corner and surface point-to-point correspondences, respectively. The terms ensure an equal weight among the point-to-point correspondences. Similarly, $\bar{\bar{C}}_{\mathcal{M}}$ is the number of point-to-mesh correspondences.

The following condition is implemented to avoid a too high influence of point-to-mesh



(a) Loss functions for nonlinear least-squares optimization using a scaling factor of $\alpha = 1.0$

(b) Huber loss function for different scaling parameters α

Figure 4.6: Illustration of the most common loss functions for nonlinear least-squares optimization (a) and different scaling parameters for the Huber loss function (b).

correspondences in the map optimization, when the number of mesh features is low:

$$\bar{\bar{C}}_{\mathcal{M}} = \begin{cases} 100 & \bar{\bar{C}}_{\mathcal{M}} < 100 \\ \bar{\bar{C}}_{\mathcal{M}} & \bar{\bar{C}}_{\mathcal{M}} \geq 100 \end{cases} \tag{4.3}$$

The residuals are Euclidean distances, i.e., depicted as the red lines in Fig. 4.5. The cost functions, computing the residual r, are wrapped inside a loss function $\rho(\cdot)$, forming a residual block. It takes the squared residual (squared norm $||r||_2^2$) as input.

A variety of loss functions exist in the literature [123]. A selection is visualized in Fig. 4.6(a). The trivial loss function maps the input residual to the output, whereas the Huber, SoftLOne, and Cauchy loss functions use a more complex mapping function. The main goal of the loss function is to reduce the influence of outliers in the overall optimization process, i.e., a wrong correspondence with a high point-to-point or point-to-mesh distance will result in a high residual and therefore heavily influence the optimization. A wrong correspondence might be due to a LiDAR scan point not belonging to the reference object, which survived the scan isolation step. This may prevent a proper convergence to the correct minimum. The loss function reduces this influence of high residuals. A scaling factor α can be applied to the loss function. By default, the LOAM algorithm [12] employs the Huber loss function of the Ceres solver [123] with $s = r^2$:

$$\rho(s) = \begin{cases} s & s \leq \alpha^2 \\ 2\alpha\sqrt{s} - \alpha^2 & s > \alpha^2 \end{cases} \tag{4.4}$$

Fig. 4.6(b) shows the Huber loss function for different scaling parameters $\alpha = [0.05, \cdots, 1.0]$. It can be seen that until $s \leq \alpha^2$, $\rho(r^2)$ has a linear increase, following the condition for $s \leq \alpha^2$ in Eq. 4.4. For $s > \alpha^2$, the curve flattens faster for smaller α. Generally, there is no right or wrong choice for a loss function and also for the scaling parameter. However, choosing the trivial loss function might render the optimization fully exposed to outlier residuals. In the following, the Huber loss function is chosen as default with $\alpha = 0.1$ to reduce the influence of wrong correspondences.

In fact, the loss function $\rho(\cdot)$ returns a loss vector composed of $\boldsymbol{l} = [\rho(s), \rho'(s), \rho''(s)]$, where $\rho'(s)$ and $\rho''(s)$ are the first and second derivatives of the loss $\rho(s)$, respectively. The derivatives and the Jacobian matrix are required for the gradient-based optimization solver. The first and second derivatives for the Huber loss as implemented in the Ceres solver [123] and used in this thesis are described as:

$$\rho'(s) = \begin{cases} 1 & s \leq \alpha^2 \\ \max(\varepsilon, \frac{\alpha}{\sqrt{s}}) & s > \alpha^2 \end{cases} \tag{4.5}$$

$$\rho''(s) = \begin{cases} 0 & s \leq \alpha^2 \\ -\frac{\rho'(s)}{2s} & s > \alpha^2 \end{cases} \tag{4.6}$$

A lambda variable λ was added to the normalization term of point-to-mesh correspondences in Eq. 4.1. This makes it possible to control the influence of the reference object-based relative localization throughout the map optimization process. $\lambda = 0$ is equal to only running the conventional LOAM with point-to-point correspondences since point-to-mesh correspondences are fully ignored in the optimization. A very high value of λ makes the optimization only rely on point-to-mesh correspondences. This is not recommended, since the reference object may not be visible at all times or the LiDAR scan of the reference object might be ambiguous and not unique. For example, a vertical scan slice of the fuselage of the airplane might be identical to one recorded several meters apart. Hence, a trade-off between conventional point-to-point and the proposed point-to-mesh correspondences is required.

Empirical observations have shown that an increasing value for λ with the number of map optimization iterations leads to the best result. Hence, the following frame parameters are identified: $\lambda_{\text{min}} = 0.1$ for the first iteration ($i = 1$) and $\lambda_{\text{max}} = 40$ after 35 iterations ($i = 35$). Therefore, the first map optimization iteration is mostly determined by point-to-point correspondences, whereas later iterations mainly rely on point-to-mesh correspondences. The function, which maps the lambda values for the iterations $2 - 34$ however, can vary significantly. Figure 4.7 illustrates λ as a function of the current iteration index $i \in \boldsymbol{i} = \{1, 2, \cdots, 35\}$ using linear, sigmoid and logarithmic functions. The most straightforward function performs a linear mapping:

$$\lambda_{\text{Lin}}(i) = \lambda_{\text{min}} + \frac{(i-1)(\lambda_{\text{max}} - \lambda_{\text{min}})}{\max(\boldsymbol{i}) - 1} \tag{4.7}$$

A Sigmoid function has an "S-shaped" curve and is often used as an activation function in Artificial Neural Networks (ANNs). It has a lower value than the linear function until its midpoint c, but a higher one after. λ_{Sig} using the Sigmoid function for an iteration i with a

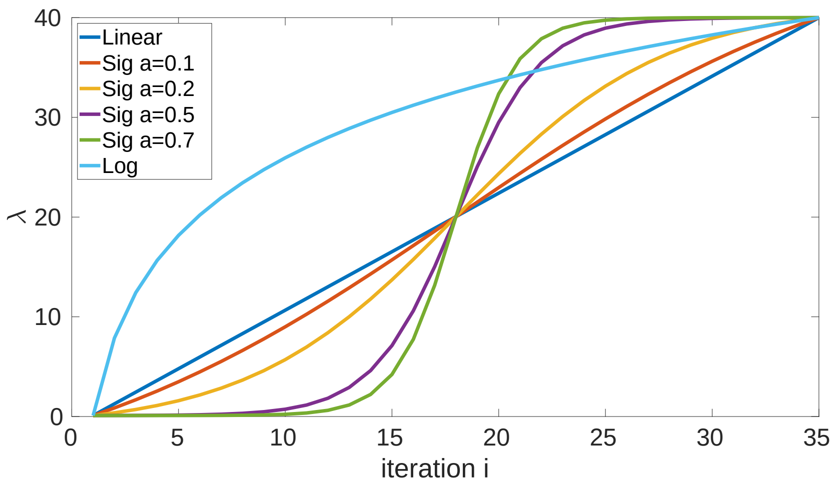


Figure 4.7: Selected functions for λ, scaled between $\lambda_{\text{min}} = 0.1$ for iteration $i = 1$ and $\lambda_{\text{max}} = 40$ at iteration $i = 35$

scaling factor a can be obtained with

$$\begin{aligned} s(i,a) &= \frac{1}{1+e^{-a(i-1-c)}}, \\ \text{with } c &= \frac{\max(\boldsymbol{i})}{2} \\ \lambda_{\text{Sig}}(s(i,a)) &= \lambda_{\text{min}} + \frac{(s(i,a) - s(\min(\boldsymbol{i}),a))(\lambda_{\text{max}} - \lambda_{\text{min}})}{s(\max(\boldsymbol{i})+1,a) - s(\min(\boldsymbol{i}),a)} \end{aligned} \tag{4.8}$$

Function $s(i,a)$ computes the Sigmoid value for iteration i with scaling factor a. Function $\lambda_{\text{Sig}}(s(i,a))$ then scales the Sigmoid value for use as lambda weight.

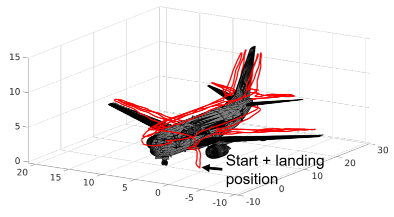
In a similar fashion, a scaled logarithmic curve can be generated for the iteration $i \in \boldsymbol{i}$:

$$\lambda_{\text{Log}}(i) = \lambda_{\text{min}} + \frac{(\log(i) - \log(\min(\boldsymbol{i})))(\lambda_{\text{max}} - \lambda_{\text{min}})}{\log(\max(\boldsymbol{i})) - \log(\min(\boldsymbol{i}))} \tag{4.9}$$

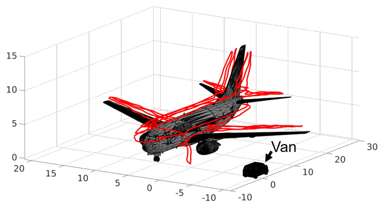
The influence of the different functions for lambda on the SLAM performance will be evaluated in the experimental results section of this chapter.

4.3 Experimental setup

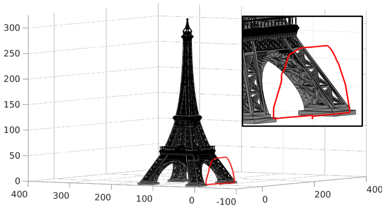
The Gazebo simulation environment is used to generate three datasets of visual inspection scenarios. Figure 4.8 visualizes the followed trajectory by a UAV for each of the scenarios. Scenario 1 (a) uses the airplane, Scenario 2 (b) the van, and Scenario 3 (c) the Eiffel Tower as the reference object. Scenarios 1 and 2 are indoors inside a hangar, while Scenario 3 is outdoors. Scenario 2 only uses the small van as the reference object. Therefore, it is not visible at all times in the LiDAR scans, especially when the UAV is on the other side of the airplane. Also, due to the small size of the van compared to the airplane as a reference object, a lot less point-to-mesh correspondences can be extracted, making the reference object-based relative localization very challenging.



(a) Scenario 1 - Airplane



(b) Scenario 2 - Van



(c) Scenario 3 - Eiffel Tower

Figure 4.8: Illustration of the three simulated visual inspection scenarios and the followed trajectory in red. Scenario 1 (a) uses the airplane, Scenario 2 (b) the van, and Scenario 3 (c) the Eiffel Tower as the reference object. Adapted from [1], ©2021 IEEE.

Table 4.1: Metadata for the generated datasets. For each scenario, two datasets were generated. One using a VLP-16 and one using an OS1-128 LiDAR sensor. Adapted and modified from [1], ©2021 IEEE.

dataset	scenario	LiDAR	#scans	avg. vel.	length	duration
1	1	VLP-16	15111	0.35 m/s	514 m	25m 11s
2	1	OS1-128	9905	0.48 m/s	474 m	16m 30s
3	2	VLP-16	9718	0.49 m/s	474 m	16m 11s
4	2	OS1-128	9726	0.49 m/s	474 m	16m 15s
5	3	VLP-16	5674	0.51 m/s	291 m	9m 26s
6	3	OS1-128	5762	0.51 m/s	291 m	9m 35s

The robotic platform is a UAV and is identical to the one described in Section 3.3. Here too, a Velodyne VLP-16 LiDAR sensor with added zero-mean Gaussian noise with $\sigma = 0.03$ for ranging errors is employed. Besides, for each scenario, an Ouster OS1-128 LiDAR sensor is simulated, following the manufacturer specifications with ranging errors of $\sigma = 0.05$. Hence, two datasets are generated for each of the scenarios using either of the LiDAR sensors. Both types of LiDAR sensors operate at 10 Hz in all datasets. As described in Section 2.1.2, the OS1-128 features 128 scan lines compared to 16 of the VLP-16 and generates up to 2.6 million points per second. The OS1-128 is the current high-end of spinning LiDAR sensors in the industry with its vertical scan line density. It is added to the experiments to further evaluate the performance of the approach, in case the data generated by the VLP-16 is insufficient.

Table 4.1 shows the metadata for each recorded dataset. In Dataset 1, the UAV was manually controlled. Datasets 2-4 follow the same trajectory but with an autonomous flight controller. Hence the difference in #scans for the datasets and the slower average velocity in Dataset 1. The UAV of the Datasets 5 and 6 also used an autonomous flight controller. It was configured with an average speed of 0.5 m/s, as can be seen in the average velocity of the Datasets 2-6.

The LiDAR sensor is continuously actuated back and forth to increase the FoV. Note that despite the same trajectory being followed in the Datasets 1 and 2, the LiDAR scans are sub-

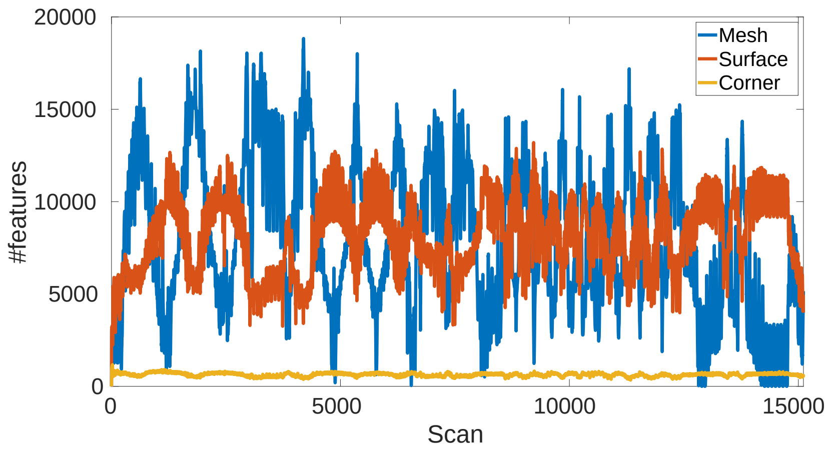


Figure 4.9: Illustration of the number of mesh, surface, and corner features for each LiDAR scan of Dataset 1

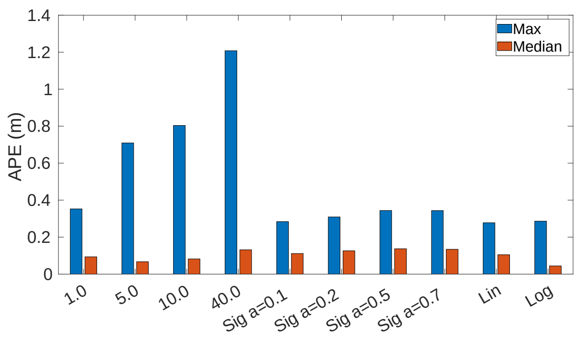
stantially different due to the varying LiDAR actuation. Hence, each dataset recording will yield different LiDAR scans despite identical trajectories.

4.4 Experimental results

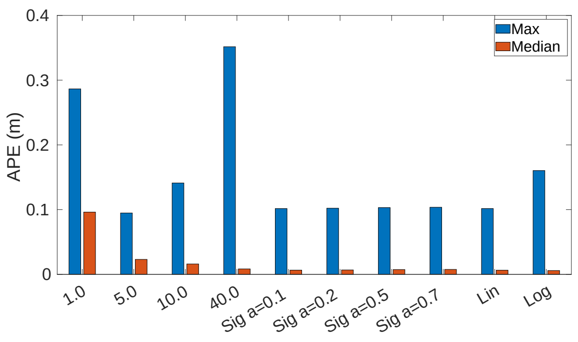
LOAM and R-LOAM are capable of running online, e.g., the algorithms try to cope with the rate of incoming LiDAR scans by dropping frames. The results presented in this section are generated in offline mode. Here, each LiDAR scan of the dataset is fully processed to achieve reproducible results.

4.4.1 Scenario 1 - Airplane as a reference object

Due to the LiDAR rotation and position relative to the reference object, the airplane might not be visible at all times in the LiDAR scans. Figure 4.9 shows the number of mesh, surface, and corner features extracted in each LiDAR scan of Dataset 1. While the number of mesh and surface features varies significantly, it is quite stable for the corner features. In fact, the surface and corner features were denoted as $\mathcal{P}_{\mathcal{H}_{\text{less}}}$ and $\mathcal{P}_{\mathcal{E}_{\text{less}}}$ previously. Hence, they represent the less flat and less sharp features, respectively. Only these are sent to the mapping module. The number of less sharp corner features is stable because a fixed number is extracted from each scan section and scan line (see Section 2.1.4.1). All remaining features are assigned to the less flat category. Therefore, a significant difference in corner and surface features can be seen. The number of mesh features is highly dependent on the visibility of the reference object. Especially at the end of the trajectory of Dataset 1, when the UAV is flying above the airplane, very few and sometimes no mesh features are extracted. Since the LiDAR sensor is mounted horizontally, only when it reaches a tilted endpoint (i.e., $\pm 0.6\,\mathrm{rad}$), the LiDAR scan contains some points on the airplane surface. When no mesh features are extracted in a scan, the map optimization only relies on point-to-point correspondences and is identical to the conventional LOAM algorithm.



(a) 2 iterations



(b) 35 iterations

Figure 4.10: APE (m) as a function of different lambda values for Dataset 1 using R-LOAM. The results are shown for 2 iterations (a) and 35 iterations (b). $[1.0, 40.0]$ are constant values for λ, whereas the others are functions of the current iteration as shown in Fig. 4.7.

Figure 4.10 shows the results when using different λ-functions using R-LOAM on Dataset 1. As previously discussed, it controls the weight of point-to-mesh correspondences in the joint optimization formulation. In (a) it can be seen, that a value of $\lambda = 5.0$ yields the lowest median APE among the constants. However, the max. APE increases with increasing constant lambda. In comparison, the Sigmoid and linear functions show a lower max. APE, but a slightly increased median APE. The logarithmic function yields the best results with one of the lowest max. and median APEs. When looking at the results of the Sigmoid, linear, and logarithmic functions it should be noted, that these are only for two iterations (which is the default for LOAM). Hence, only two lambda values were sampled from the functions at iteration 1 and 2. In fact, $\lambda = 0.1$ for all non-constant functions in the first iteration. Only the second λ value depends on the actual function. The Sigmoid function with $a = 0.1$ is almost linear, therefore the result is nearly identical to the linear function. In contrast, $a = 0.7$ yields very low λ in the first two iterations as can be seen in Fig. 4.7. The lower λ, the closer the results get to the conventional LOAM performance. The logarithmic function starts with $\lambda = 0.1$, while the second value is $\lambda > 7$. This has proven to be most suitable for two iterations. A high lambda value in the first iteration even worsens the performance, as can be seen for the constant $\lambda = 5.0$ and $\lambda = 10.0$ results. While a lower constant lambda achieved better performance for 2 iterations, the opposite can be seen for 35 iterations in Fig. 4.10(b). A high constant lambda, e.g., 40, yields the lowest median APE, but the highest max. APE at the same time. The Sigmoid functions perform very similarly for all parameters a. The logarithmic function achieves the lowest median APE with only $0.6\,\mathrm{cm}$ at the cost of a slightly higher max. APE.

Considering the results for 2 and 35 iterations, the logarithmic function for lambda is applied to all of the following experiments.

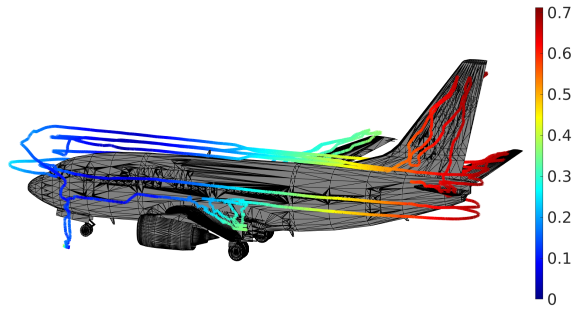
Table 4.2 shows the results for Scenario 1 for LOAM and R-LOAM when using a different number of iterations for correspondence estimation and map optimization by means of APE (cm) and RE (deg). In this scenario, the airplane is used as a reference object by R-LOAM. By default, the LOAM algorithm performs two iterations to maintain the capability to run on-

Table 4.2: Experimental results for Scenario 1 with a varying number of iterations. The best results for each dataset are marked in **bold**. Adapted and modified from [1], ©2021 IEEE.

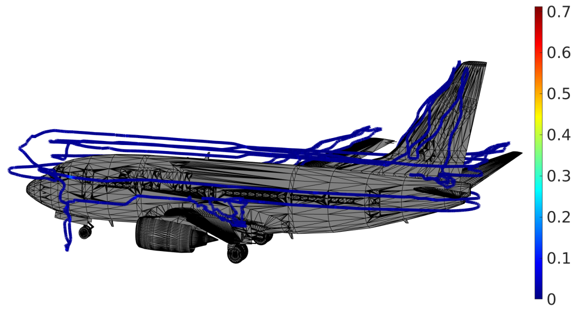
	method	#iter	APE in cm			RE in deg		
			max	mean	median	max	mean	median
VLP-16 (Dataset 1)	LOAM [12]	2 (def)	102.4	55.8	50.5	2.98	2.19	2.20
		5	116.9	64.7	57.7	3.13	2.55	2.56
		15	101.0	55.5	51.1	3.14	2.11	2.12
		25	69.6	35.9	30.9	3.14	1.58	1.59
		35	71.2	36.8	31.9	3.14	1.61	1.61
	R-LOAM	2 (def)	28.6	4.6	4.4	**0.88**	0.22	0.22
		5	23.2	4.0	4.0	1.83	0.32	0.31
		15	19.5	3.3	3.1	1.67	0.26	0.25
		25	17.0	1.3	1.1	1.38	0.11	0.11
		35	**16.0**	**0.8**	**0.6**	0.94	**0.04**	**0.04**
OS1-128 (Dataset 2)	LOAM [12]	2 (def)	20.8	12.7	13.2	0.29	0.10	0.09
		5	20.9	13.0	13.5	0.29	0.09	0.08
		15	20.9	13.0	13.5	0.29	0.09	0.08
		25	20.9	13.0	13.5	0.29	0.09	0.08
		35	20.9	13.0	13.5	0.29	0.09	0.08
	R-LOAM	2 (def)	18.3	2.9	2.6	0.44	0.12	0.13
		5	13.8	2.1	1.9	0.32	0.09	0.09
		15	10.0	1.6	1.4	0.29	0.08	0.08
		25	7.8	1.4	1.3	0.29	**0.07**	**0.07**
		35	**6.3**	**1.3**	**1.2**	**0.28**	**0.07**	**0.07**

line. The results are shown until 35 iterations since a higher number did not show significant changes anymore once the full convergence has been reached. It can be seen that R-LOAM outperforms LOAM in both datasets for the VLP-16 and the OS1-128 LiDAR sensor. A higher number of iterations was always beneficial for R-LOAM, however, less significant when using the OS1-128 LiDAR sensor. The overall lowest median APE was achieved by R-LOAM at 35 iterations with the VLP-16 sensor ($0.6\,\mathrm{cm}$) and the lowest max. APE by R-LOAM at 35 iterations with the OS1-128 sensor ($6.3\,\mathrm{cm}$). The higher number of points in each scan of Dataset 2 results in faster convergence. This can be clearly seen for LOAM at $5 - 35$ iterations. The scans already fully converge after only 5 iterations, without the chance for further improvement. On the other hand, when using R-LOAM with the additional point-to-mesh correspondences, a further reduction to 35 iterations can be achieved.

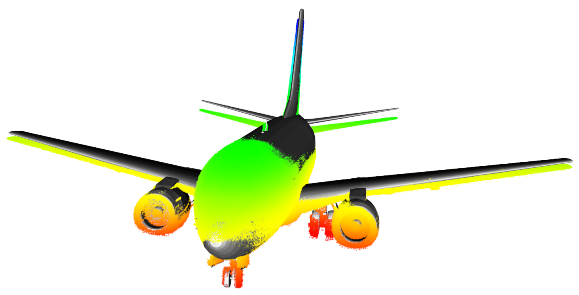
Figures 4.11(a) and (b) illustrate the APE mapped onto the trajectory of Dataset 1 after 35 correspondence estimation and map optimization iterations. LOAM suffers from drift towards the rear of the airplane, even after 35 iterations. The reason is mainly inaccurate pose



(a) APE (m) using LOAM [12]



(b) APE (m) using R-LOAM


(c) Dense 3D reconstruction of (a)

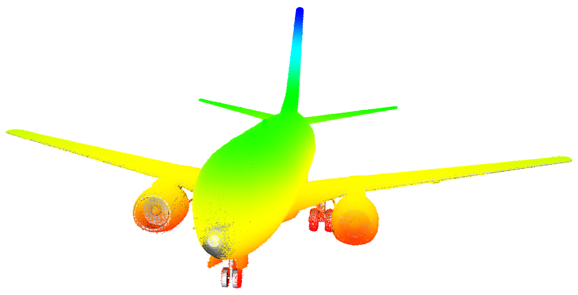

(d) Dense 3D reconstruction of (b)

Figure 4.11: Visualization of the APE mapped onto the trajectory of Dataset 1 for LOAM (a), and R-LOAM (b) after 35 iterations. (c) and (d) show the dense 3D reconstruction and the GT mesh, respectively.

estimation and accumulated scan insertion with the wrong transform in the mapping module. The drift recovers towards the front since the UAV flies closer to the position, where the map was created, i.e., the world frame $\{W\}$. As shown previously in Table 4.2, R-LOAM has a median APE of $0.6\,\mathrm{cm}$ and no drift can be seen. Figures 4.11(c) and (d) show the dense 3D reconstruction using the poses of (a) and (b), respectively. The point clouds are shown as a GT mesh overlay. The reconstruction is created by accumulating the raw LiDAR scans with the corresponding map-optimized transforms. For LOAM, the drift can be clearly seen towards the airplane's rear. The vertical stabilizer of the airplane is mapped below its actual position. Also, the wings are mapped too low. The point cloud of the fuselage disappears towards the rear into the GT mesh, indicating the drift. In comparison, the dense 3D reconstruction (d) using the poses of R-LOAM perfectly aligns with the GT mesh. The joint optimization with point-to-mesh correspondences guides the localization and mapping process of R-LOAM, reducing trajectory error and improving map quality.

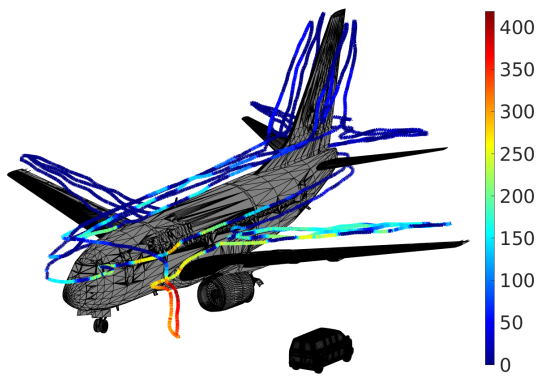
4.4.2 Scenario 2 - Van as a reference object

Table 4.3 shows the results for Scenario 2. In this scenario, R-LOAM uses the van as a reference object. However, the trajectory is identical to Scenario 1. First, it can be seen that the performance of LOAM is quite different compared to Table 4.2. Despite the same trajectory, the LiDAR scans differ significantly due to the varying LiDAR actuation with the gimbal. Hence, each dataset recording will yield different results. Interestingly, when using the VLP-

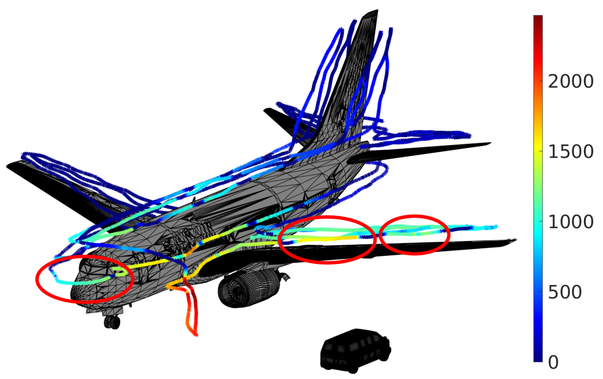
Table 4.3: Experimental results for Scenario 2 with varying number of iterations. The best results for each dataset are marked in **bold**. Adapted and modified from [1], ©2021 IEEE.

	method	**#iter**	**APE in** cm			**RE in** deg		
			max	mean	median	max	mean	median
VLP-16 (Dataset 3)	LOAM [12]	2 (def)	37.7	22.2	19.0	1.46	0.84	0.83
		5	136.7	72.7	65.0	3.24	3.08	3.08
		15	60.5	28.9	24.0	3.52	1.65	1.63
		25	59.9	28.6	23.8	3.52	1.63	1.62
		35	60.4	28.9	23.9	3.52	1.64	1.62
	R-LOAM	2 (def)	39.7	20.0	17.9	**1.35**	0.82	0.81
		5	16.4	7.5	7.3	1.44	0.44	0.44
		15	**15.3**	**7.0**	**6.3**	1.42	**0.37**	**0.37**
		25	23.5	13.2	13.5	1.41	0.38	**0.37**
		35	23.6	13.2	13.5	1.41	**0.37**	**0.37**
OS1-128 (Dataset 4)	LOAM [12]	2 (def)	23.9	14.5	14.8	0.46	0.08	0.07
		5	24.2	15.0	15.3	0.41	0.08	0.06
		15	24.1	15.1	15.4	0.41	0.07	0.06
		25	24.1	15.1	15.4	0.41	0.07	0.06
		35	24.1	15.1	15.4	0.41	0.07	0.06
	R-LOAM	2 (def)	32.6	6.3	6.3	0.73	0.10	0.09
		5	15.2	5.2	4.9	0.40	0.07	0.06
		15	**12.3**	4.8	4.3	**0.39**	**0.06**	**0.05**
		25	13.5	**4.7**	**4.2**	**0.39**	**0.06**	**0.05**
		35	14.2	**4.7**	**4.2**	**0.39**	**0.06**	**0.05**

16 (Dataset 3) for LOAM, two iterations perform best. A higher number converges to wrong minima in the optimization process, as can be seen in the sudden increase for five iterations. Despite the small van, R-LOAM performs better than LOAM with the lowest median APE at 15 iterations with 6.3 cm. Also here, a higher number of iterations increases the APE and the median APE converges to 13.5 cm. When looking at the results of Dataset 4, one notices that similar to Scenario 1, LOAM fully converges after around 5 iterations. The benefit of using the OS1-128 for the van as a small reference object becomes clearly visible for R-LOAM. The median APE at 35 iterations improves by almost 10 cm for R-LOAM between Dataset 3 and Dataset 4 compared to a slight decrease between Dataset 1 and Dataset 2. Therefore it can be concluded, that using a LiDAR sensor with more scan points and more vertical scan lines can indeed improve R-LOAM performance, if the reference object is small, or the distance is large. Similar to Scenario 1, R-LOAM also achieves the lowest RE in Scenario 2. Due to the distance and the size of the reference object, only 22 % of the scans using the VLP-16 (Dataset 3) have more than 100 mesh features in comparison to the OS1-128 (Dataset 4) with



(a) VLP-16



(b) OS1-128

Figure 4.12: Number of extracted mesh features mapped onto the trajectory of Scenario 2 using a VLP-16 (a) and an OS1-128 (b). Generally, more mesh features can be extracted with the OS1-128 due to the higher number of scan points. The red circles show areas with a higher ratio of mesh features when using an OS1-128 compared to a VLP-16.

$45\,\%$.

Figure 4.12 visualizes the number of extracted mesh features for Scenario 2 using the VLP-16 (a) and OS1-128 4.12(b) LiDAR sensors. Due to the higher number of scan points, more mesh features can be extracted with an OS1-128. When the UAV is still close to the van as a reference object, the highest number of mesh features can be extracted. In the back of the airplane and especially on the other side of the fuselage, the van is not visible and the R-LOAM algorithm has to rely on conventional point-to-point correspondences. The red circles in Fig. 4.12(b) show areas where a higher ratio of mesh features can be extracted compared to a VLP-16. This is likely due to the increased vFoV of the OS1-128 as was shown in Section 2.1.2.

4.4.3 Scenario 3 - Eiffel Tower as a reference object

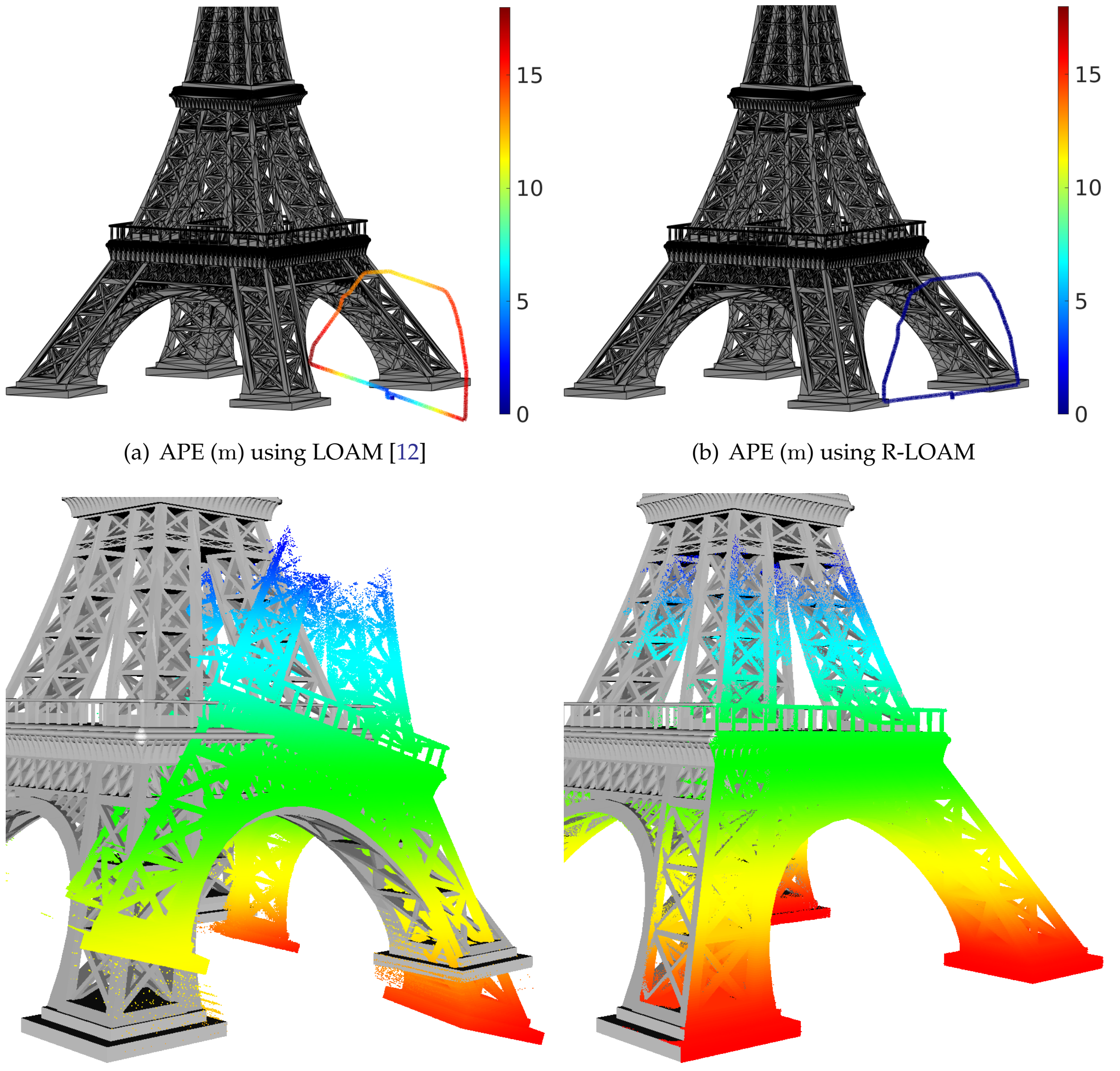
Table 4.4 shows the results for Scenario 3. Here, the Eiffel Tower is used as a reference object for R-LOAM. LOAM fails with the VLP-16, regardless of the number of iterations. At 15 iterations a sudden increase in APE can be seen. This is most likely due to wrong point-to-point correspondence estimation using point features. A single wrongly inserted scan can deteriorate the map quality significantly and therefore have a negative impact on future localization and mapping. For R-LOAM, point-to-point correspondences are also affected by wrong scan insertions. However, it does not affect the point-to-mesh correspondences, which are jointly optimized. Hence, using a tightly coupled reference object, whose quality is not deteriorating over time, can compensate for wrong scan insertions into the map. R-LOAM fails at two iterations but converges to a very low error of $< 2.4\,\mathrm{cm}$ median APE at ≥ 5 iterations. At ≥ 15 iterations, R-LOAM achieves a subcentimeter median APE. The median RE amounts to only $0.01\,\mathrm{deg}$ at 35 iterations. Using the OS1-128 sensor, LOAM converges well for > 15 iterations to a minimum of $9.3\,\mathrm{cm}$ median APE. For R-LOAM, two iterations are sufficient to achieve $6.4\,\mathrm{cm}$ median APE. With this sensor, the max. APE is as low as $7.3\ \mathrm{cm}$ for R-LOAM

Table 4.4: Experimental results for Scenario 3 with varying number of iterations. The best results for each dataset are marked in **bold**. Adapted and modified from [1], ©2021 IEEE.

	method	#iter	APE in cm			RE in deg		
			max	mean	median	max	mean	median
VLP-16 (Dataset 5)	LOAM [12]	2 (def)	1512.2	495.0	499.3	10.09	3.66	2.47
		5	443.4	327.0	380.2	5.35	4.38	4.08
		15	1797.5	1244.7	1420.9	17.56	17.25	17.28
		25	1696.2	1166.6	1326.4	16.61	16.36	16.40
		35	1800.5	1218.6	1338.5	18.42	18.17	18.21
	R-LOAM	2 (def)	1869.5	226.0	31.5	17.01	2.73	0.29
		5	88.2	5.1	2.4	1.60	0.07	0.04
		15	72.9	1.2	0.5	**0.32**	**0.02**	0.02
		25	61.5	0.7	**0.3**	0.40	**0.02**	0.02
		35	**48.9**	**0.5**	**0.3**	0.34	**0.02**	**0.01**
OS1-128 (Dataset 6)	LOAM [12]	2 (def)	2472.0	1109.7	1058.2	12.77	6.12	4.97
		5	1063.1	218.2	105.0	9.23	2.70	1.49
		15	47.8	10.2	9.3	1.01	0.14	0.14
		25	45.7	10.8	10.5	0.94	0.12	0.10
		35	39.9	11.5	11.3	0.40	0.11	0.11
	R-LOAM	2 (def)	94.1	11.1	6.4	1.42	0.19	0.12
		5	42.9	3.5	2.4	0.69	0.10	0.08
		15	13.1	1.6	1.2	**0.25**	0.08	**0.07**
		25	8.5	1.3	1.1	**0.25**	0.08	**0.07**
		35	**7.3**	**1.2**	**1.0**	**0.25**	**0.07**	**0.07**

at 35 iterations. Also in this scenario, using a LiDAR with more vertical scan lines and points significantly reduces the APE and RE for LOAM and R-LOAM.

Figures 4.13(a) and (b) show the APE mapped onto the trajectory of Dataset 5 for LOAM and R-LOAM after 35 iterations, respectively. Early wrong scan matches create a tilted map when using LOAM. This leads to a high APE throughout the whole trajectory. The tilted 3D reconstruction in Figure 4.13(c) visualizes the wrong map representation. On the left, a few scans can be seen, which were correctly inserted. However, once several scan features were inserted into the map with the wrong transform, its quality deteriorates rapidly. In this case, the Eiffel Tower representation manifested after a few wrong scan insertions, as can be seen in the still complete reconstruction. Since LOAM does not have an absolute reference, the algorithm continues with the tilted map. In contrast, R-LOAM uses the triangular mesh of the Eiffel Tower as an absolute reference object. Early wrong scan insertions are prevented and the 3D reconstructed point cloud perfectly aligns with the GT mesh. The mean APE amounts to $0.5\,\mathrm{cm}$ for R-LOAM on this dataset.



(a) APE (m) using LOAM [12]

(b) APE (m) using R-LOAM

(c) Dense 3D reconstruction of (a)

(d) Dense 3D reconstruction of (b)

Figure 4.13: Visualization of the APE mapped onto the trajectory of Dataset 5 for LOAM (a), and R-LOAM (b) after 35 iterations. (c) and (d) show the dense 3D reconstruction and the GT mesh, respectively. The point cloud color is height-coded.

4.5 Limitations

The results presented in Section 4.4 assume an exact triangular mesh of the 3D reference object, i.e., the mesh representation is identical to the one used in the simulation to generate the LiDAR data. It is not possible to make a general statement about how deviations in the geometry influence the R-LOAM performance. This certainly depends on the extent of the deviation. It can be assumed that large deviations on a small portion of the reference object will have less influence than small deviations on the whole surface. For example, adding winglets to an airplane on the sides of the wings will cause LiDAR scan points to appear, which do not have a counterpart in the triangular mesh. Hence, these points will find point-to-mesh correspondences on the wing edges in the mesh. All the other scan points will align well on the mesh after the registration. The high residuals of the winglets will be reduced by the loss function in the optimization. Therefore, few high residual outliers due to some geometry deviation might only have a small influence on R-LOAM. Now assume opened wing flaps in reality, which are closed in the model. The flaps significantly alter the shape of the wings. LiDAR scans that include a large portion of the wings will create incorrect point-to-mesh correspondences with consistently high residuals. A proper convergence can not be expected in this case.

On the other hand, the presented experiments assume the knowledge about the exact absolute position of the reference object in the map frame. When comparing it with the absolute GT, the reference object pose error will have a direct influence on the R-LOAM output pose. However, if the pose error is low (a few centimeters), the map quality of R-LOAM may not suffer too much. The biggest effect would be seen at the beginning of the exploration: The first LiDAR scans are inserted into the map, but initially do not align well with the reference object due to the pose error. After a few scans, the ego pose estimation will slowly converge until the following LiDAR scans consistently align with the mesh. Until then, the ego pose error increases when compared to the GT, since the point-to-mesh correspondences are optimized to an incorrect position in the map frame. Once all scans align with the mesh, the map quality stabilizes. Due to this early drift, point-to-point correspondences may be wrongly estimated throughout the exploration. However, assuming a perfect triangular mesh, point-to-mesh correspondences will balance out this effect to some extent.

If the reference object pose is completely off, R-LOAM will have worse performance than LOAM. In this case, it is using incorrect prior knowledge. The same holds for wrong information about the geometry of the reference object, e.g., a van instead of an airplane.

4.6 Chapter summary

This chapter proposed a tightly coupled joint optimization approach by incorporating point-to-mesh correspondences of a known reference object in the map optimization formulation of LOAM. The approach is termed Reference-LOAM or R-LOAM. A requirement is the knowledge about the geometry and location of the reference object in the global or map frame. Conventional LOAM only uses point features and estimates the ego pose with point-to-point correspondences between scan and map features. The proposed approach computes point-to-mesh correspondences between isolated scan points and virtual points on the reference object's surface. These are then added to the map optimization problem and jointly optimized with the conventional point-to-point correspondences. Normalization terms were introduced to give equal weight to corner and surface features, independently of their number. A lambda weight controls the influence of mesh features in the optimization problem.

For the experiments, the simulated UAV platform from Chapter 3 was used. The VLP-16 LiDAR sensor was replaced by an Ouster OS1-128 for some experiments. Six datasets were recorded for three scenarios using either the VLP-16 or OS1-128. Scenario 1 and 2 were indoors using an airplane and van as reference objects, while Scenario 3 was outdoors using the Eiffel Tower as a reference object.

The experimental evaluation demonstrated on Dataset 1 that a logarithmic increase for lambda is the best choice for two (default) and also 35 map optimization iterations. Two iterations offer online running capability, while 35 provide full convergence.

In all three scenarios, the OS1-128 achieved a lower APE due to its higher number of scan points and increased vFoV compared to the VLP-16. Using the airplane as a reference object, R-LOAM achieved a significantly lower APE compared to LOAM. The small van in Scenario 2 is not always visible throughout the trajectory. However, the OS1-128 LiDAR detects sufficient mesh features for R-LOAM to still show three times lower median APE compared to conventional LOAM. Scenario 3 using the Eiffel Tower is very challenging and LOAM and R-LOAM fail with the VLP-16 at only two iterations. At 35 iterations, R-LOAM achieves subcentimeter accuracy with both LiDAR sensors. Overall, R-LOAM achieved a reduction in median APE of 98.12 % for Scenario 1, 43.51 % for Scenario 2, and 99.98 % for Scenario 3 using a VLP-16 at 35 correspondence estimation and map optimization iterations compared to LOAM.

The proposed approach shows a way to directly integrate a triangular mesh into the joint optimization formulation. In practice, CAD 3D models may be available and can be used without modification. Alternatively, devices from Building Information Modeling (BIM) can create highly accurate 3D point cloud representations of the 3D reference object. These can then be converted to mesh files (see Section 2.1.3.2) or can be used directly in the point cloud format. In this case, no virtual points need to be computed, but a k-d tree is used to find correspondences. However, it should be noted that a sparse point cloud of the reference object will decrease R-LOAM performance. Depending on the size of the reference object, the computational complexity of using a point cloud as a reference object will exceed the one of a triangular mesh. For example, a straight large wall or ground can be described with a few triangles. To cover the same area with a dense point cloud, a large number of points is required as was illustrated in Fig. 2.4.

Summarizing the findings, leveraging prior knowledge about a 3D reference object can significantly improve LOAM accuracy. However, the geometry and location of the reference object must be known as accurately as possible.

Chapter 5

3D reference object-based trajectory and map optimization for LOAM

This chapter presents a novel extension to the LOAM framework [12] to enable reference object-based trajectory and map optimization in a parallelized manner on an Edge Cloud. The method is evaluated on real data captured in a visual airplane inspection scenario inside a hangar.

Parts of this chapter have been published in [2].

5.1 Problem statement

The previous chapter introduced a way to incorporate a complex 3D reference object into the conventional LOAM framework. Point-to-mesh correspondences were extracted and integrated into the map optimization formulation. While this showed significant improvements, it requires additional computational resources, which might not be available on all mobile platforms. The tightly coupled architecture makes it unsuitable for a remote SLAM setup, where the computationally expensive reference object-based relative localization can be offloaded to an Edge Cloud.

The experiments in Chapter 4 have shown the best performance of R-LOAM at higher map optimization iterations while running in offline mode, i.e., all LiDAR scans were processed without real-time constraints. This is suitable for applications where generally the best possible localization and mapping is desired, even after the exploration. One example is the capture of LiDAR scans using a UAV for visual inspection. High-resolution images are captured of the object's surface. If possible damages are found on some of these images, their location must be found to further evaluate the affected area. For this, each image must be stamped with the location it was taken. In this case, the time for improving the localization is not a constraint, but a highly accurate pose estimation is desired.

By default, LOAM operates at two iterations to maintain online capability. Optionally, this can be increased at the cost of more frame drops, assuming a constant LiDAR scan rate. In this chapter, a novel extension to the LOAM framework is proposed by loosely coupling the reference object-based relative localization to the mapping module of LOAM. The proposed setup allows for online processing of the conventional LOAM algorithm while per-

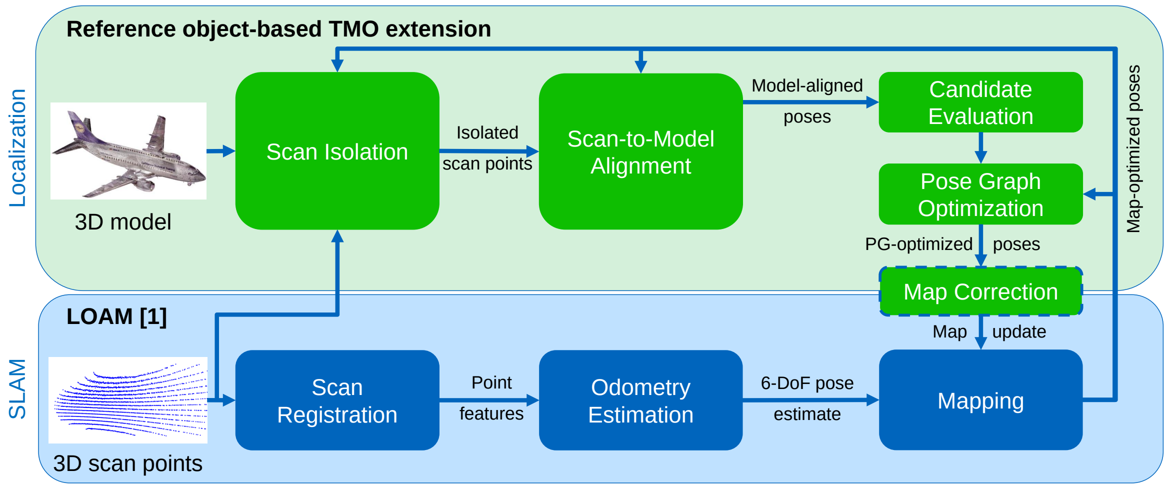


Figure 5.1: Overview of the proposed reference object-based trajectory and map optimization (TMO) pipeline. The lower part (blue) shows the conventional LOAM [12] modules and the upper part (green) shows the proposed extension. The map correction module represents the tightly-coupled interface between the extension and the LOAM algorithm. Reproduced from [2], ©2022 IEEE.

forming **T**rajectory and **M**ap **O**ptimization (TMO) in a fully parallelized manner, suitable for offloading to an Edge Cloud. Similar to the proposed method in the previous chapter, the geometry and position of the reference object in a global coordinate frame must be known. The proposed extension is termed Reference Object-LOAM (RO-LOAM).

5.2 Methodology

Figure 5.1 shows an overview of the proposed reference object-based TMO. The blue boxes show the state-of-the-art LOAM modules [12] and the green part is the proposed extension. The core idea can be divided into five steps:

1. **Scan isolation**: This part is identical to the one described in Section 4.2.2. Raw LiDAR scan points are filtered so that ideally only scan points of the reference object remain.

2. **Scan-to-model alignment**: Map-optimized poses are used as initial guess for scan-to-model alignments. A number of consecutive LiDAR scans are registered to the model, yielding refined model-aligned poses.

3. **Candidate evaluation**: Model alignments may suffer from incorrect convergence due to alignment ambiguities and therefore even increase the pose error. To determine, if the last model-aligned pose of step 2 is a candidate for TMO, the sequence of model-aligned poses is fed into an EKF and the last pose is compared to the motion prior.

4. **Pose Graph Optimization (PGO)**: If the candidate pose was verified in the previous step, a PGO is triggered to correct the poses since the last PGO. This effectively corrects the previous trajectory.

5. **Map correction**: With the corrected trajectory, the map of the LOAM algorithm can be corrected by partial rebuilding. This effectively corrects drift and positively influences the map quality for future localization and mapping.

The modules of the proposed extension are loosely coupled so that the scan isolation until the PGO can be offloaded to an Edge Cloud with a higher processing capacity. Due to this loose coupling, these modules can be connected to any LiDAR-SLAM algorithm. Only the map correction module is highly dependent on the map structure of the LiDAR-SLAM algorithm and needs to be adapted accordingly. Hence, it can be considered as an interface between the TMO extension and the LiDAR-SLAM algorithm.

For the proposed approach, the LiDAR sensor is assumed to be mounted on the robot via a gimbal. The transform of the LiDAR sensor $\{L\}$ in the robot frame $\{D\}$ is defined as ${}^{D_t}\boldsymbol{T}_{L_t}$ and can be created from the actuator readings. The forward kinematics of the LiDAR sensor in the world/map frame $\{W\}$ can then be described as

$$ {}^{W}\boldsymbol{T}_{L_t} = {}^{W}\boldsymbol{T}_{D_t}\,{}^{D_t}\boldsymbol{T}_{L_t} \,. \tag{5.1}$$

The transform ${}^{W}\boldsymbol{T}_{D_t}$ describes the pose of the robot $\{D\}$ in the world frame $\{W\}$ and is computed by the SLAM algorithm.

The blue modules in Fig. 5.1 of the conventional LOAM algorithm remain largely unmodified by the extension and are not further explained. Only the extracted scan features are transformed to the robot frame using the actuator transform. The interested reader is referred to the original paper [12] or the description in Section 2.1.4.1 and 2.2.1 for further information on LOAM.

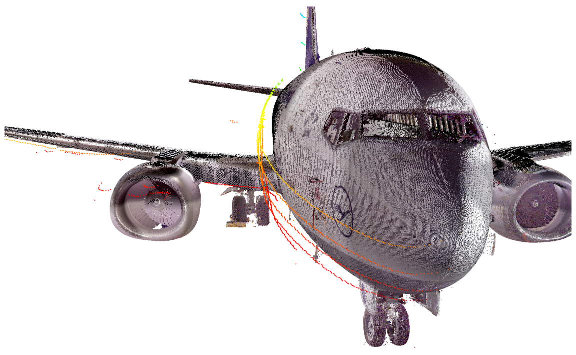
In the following, the modules of the extension are explained in detail.

5.2.1 Scan isolation

The task of the scan isolation module is to filter out points, which do not belong to the reference object. This part is essentially identical to the one described in Section 4.2.2. Incoming LiDAR scans are first transformed to the world frame utilizing the map-optimized poses of LOAM ${}^{W}\boldsymbol{T}_{D_t}$, and the actuator transform ${}^{D_t}\boldsymbol{T}_{L_t}$ in Eq. 5.1. Using a bounding box cropping method centered at the position of the reference object, the LiDAR scan is filtered. The majority of the scan points should now belong to the reference object. This step is mainly to speed up the correspondence estimation and to support the following scan-to-model alignment by reducing the influence of outliers. Alternatively, a maximum correspondence distance threshold can be used, at the cost of increased computational complexity.

5.2.2 Scan-to-model alignment

The task of the scan-to-model alignment module is to refine a map-optimized pose ${}^{W}\boldsymbol{T}_{D_t}$ to retrieve a scan-to-model aligned pose ${}^{W}\tilde{\boldsymbol{T}}_{D_t}$. For this, the LiDAR scan $\mathcal{P}^{D_t}$ captured at time t, is transformed to the world frame $\{W\}$ with the map-optimized transform ${}^{W}\boldsymbol{T}_{D_t}$ as initial guess. Figure 5.2 illustrates a LiDAR scan $\mathcal{P}^{W_t}$ transformed to the world frame before the


(a) Isolated LiDAR scan before the scan-to-model alignment. The offset is due to the error in the map-optimized pose.

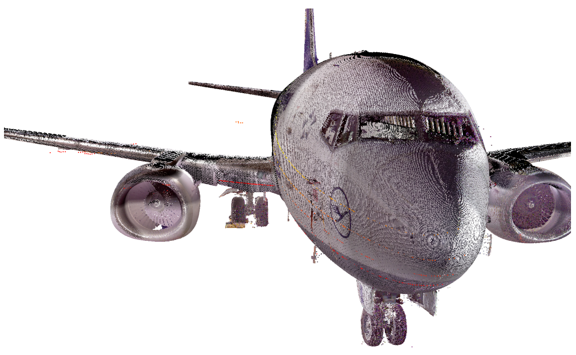

(b) Isolated LiDAR scan after 100 iterations of scan-to-model alignment. The scan is fully aligned to the model surface.

Figure 5.2: Illustration of an isolated LiDAR scan before (a) and after (b) the scan-to-model alignment. A dense point cloud of a B737 airplane was used as a reference model. The isolated LiDAR scan in (a) was transformed to the world frame with the map-optimized pose as an initial guess. After the scan-to-model alignment (b), this pose is refined to a very low error. The LiDAR scan then perfectly aligns with the model surface and has a very low MSE.

scan-to-model alignment (a) and after 100 iterations (b). This may result in a highly refined pose ${}^{W}\tilde{\boldsymbol{T}}_{D_t}$ with a possibly very low translational and rotational error.

The ICP algorithm is employed for the scan-to-model alignment process. It is formulated as a nonlinear least-squares optimization problem with the total cost J:

$$J(\boldsymbol{q}, \boldsymbol{t}) = \sum_{c \in C} \rho(\|f(c, \boldsymbol{q}, \boldsymbol{t})\|_2^2) \ . \tag{5.2}$$

$C = \{c_1, c_2, \cdots, c_l\}$ is the set of correspondences between the isolated LiDAR scan with l points and the points of the reference object point cloud. Hence, a correspondence $c = \langle p^L, p^W_{\mathcal{M}} \rangle$ is a tuple, consisting of an isolated LiDAR scan point p^L and a point $p^W_{\mathcal{M}}$ of the reference model $\mathcal{M}$. Assuming the reference object model is in the form of a point cloud, a *k*-d tree can be used to efficiently find the correspondences with a nearest neighbor search. Here, correspondences with high distances can be filtered with a maximum correspondence distance threshold. The cost function $f(\cdot)$ is formulated as:

$$f(c, \boldsymbol{q}, \boldsymbol{t}) = (\boldsymbol{q} p^L \boldsymbol{q}^{-1} + \boldsymbol{t}) - p^W_{\mathcal{M}} \text{ , with } c = \langle p^L, p^W_{\mathcal{M}} \rangle \ . \tag{5.3}$$

The isolated scan point p^L is continuously transformed to the world frame with the optimized quaternion $\boldsymbol{q}$ and translational vector $\boldsymbol{t}$. The residual is the Euclidean distance to the corresponding point $p^W_{\mathcal{M}}$ in the reference model point cloud. The squared residual is wrapped inside a Huber loss function, diminishing the effect of outliers with high residuals. The total cost J of all residuals in Eq. 5.2 is minimized with a Levenberg-Marquardt trust-region algorithm [77], [78].

Before the scan-to-model alignment, the isolated scan point cloud can be downsampled in order to speed up the correspondence estimation and scan-to-model alignment. The steps of correspondence estimation and optimization can be repeated for many more iterations than the joint map optimization of R-LOAM in Chapter 4. This is due to the offloaded and fully parallelized process, which is independent of the LOAM processes running online on the robot. The optimal number of iterations and downsampled isolated scan points are determined in the experimental section of this chapter.

The scan-to-model alignment may fail due to a wrong convergence, inaccurate initial guess, or simply alignment ambiguities. For example, the LiDAR scan may align very well in different sections of the reference model, resulting in a very low MSE. To verify that the alignment converged to the correct solution, a motion prior filtering step is employed. A sequence of scan-to-model aligned poses is used to check if the last pose follows a motion model.

The length of the sequence used for the verification is specified by a parameter M. Hence, $M + 1$ isolated LiDAR scans $\mathcal{P}^{D_{t-M:t}}$ captured at times $t - M$ until t are registered to the reference model in a parallelized manner using their respective initial guesses (map-optimized poses) ${}^{W}\boldsymbol{T}_{D_{t-M:t}}$. The parameter L controls the frequency of scan-to-model alignments and therefore TMO attempts. After L map-optimized poses have been buffered, the parallelized scan-to-model alignment of the most recent $M + 1$ isolated LiDAR scans is triggered. Note that after a successful attempt, the buffer is reset until again L isolated LiDAR scans have been buffered. The buffer is not reset, if the attempt failed, e.g., by not passing the candidate evaluation step in the following section (5.2.3). Instead, another scan-to-model alignment is triggered immediately, if the scan sequence is not identical (i.e., new scans have been received in the meantime). A scan-to-model alignment is also only triggered if each of the isolated LiDAR scans $\mathcal{P}^{D_{t-M:t}}$ has more than 50 points. Otherwise, this sequence is skipped and the module waits for new incoming scans and corresponding map-optimized poses.

The ultimate output of the scan-to-model alignment module are the refined poses ${}^{W}\tilde{\boldsymbol{T}}_{D_{t-M:t}}$. ${}^{W}\tilde{\boldsymbol{T}}_{D_t}$ is the last scan-to-model aligned pose and resembles the candidate for TMO. The pose sequence is only sent to the following evaluation module, if the MSE of the candidate pose ${}^{W}\tilde{\boldsymbol{T}}_{D_t}$ after the convergence is < 0.001. If the MSE is higher, it is rather unlikely that the scan-to-model alignment was successful and has a low error. In this case, the current attempt is dropped and the next sequence with new scans is processed.

5.2.3 Candidate evaluation

As described previously, the scan-to-model alignment may not converge to a good solution with a low pose error, despite a low MSE. This is due to alignment ambiguities if the scan slice of the reference object is not unique. To find out if the alignment process was successful, the sequence of scan-to-model alignment poses from the previous step is evaluated. If the last scan-to-model aligned pose ${}^{W}\tilde{\boldsymbol{T}}_{D_t}$ (= candidate pose for TMO) follows the motion model of the previous scan-to-model aligned poses ${}^{W}\tilde{\boldsymbol{T}}_{D_{t-M:t-1}}$, the alignment is considered successful.

An EKF is leveraged to verify the candidate against a motion prior. The standard EKF formulation of a prediction step

$$\begin{aligned} \boldsymbol{x}_{K|K-1} =& f(\boldsymbol{x}_{K-1|K-1}) \\ \boldsymbol{P}_{K|K-1} =& \boldsymbol{F}_{K-1}\boldsymbol{P}_{K-1|K-1}\boldsymbol{F}_{K-1}^{T} + \boldsymbol{Q}_{K-1} \end{aligned} \tag{5.4}$$

and correction step is used:

$$\boldsymbol{K}_K = \frac{\boldsymbol{P}_{K|K-1}\boldsymbol{H}_K^T}{\boldsymbol{H}_K\boldsymbol{P}_{K|K-1}\boldsymbol{H}_K^T + \boldsymbol{R}_K} \tag{5.5}$$

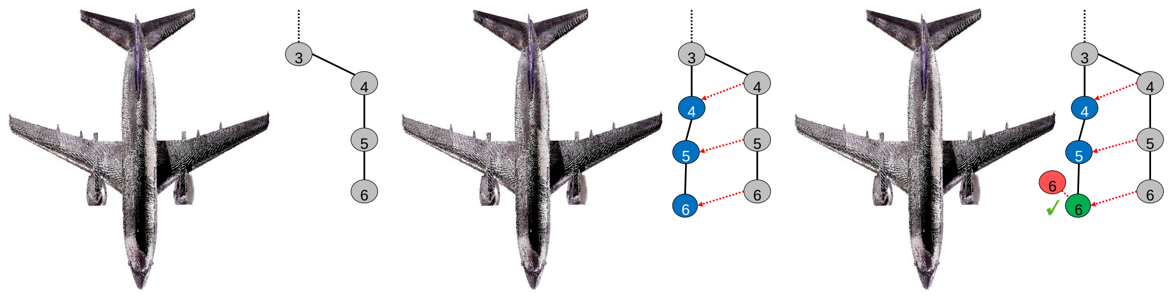
$$\begin{aligned} \boldsymbol{x}_{K|K} =& \boldsymbol{x}_{K|K-1} + \boldsymbol{K}_K(\boldsymbol{z}_K - \boldsymbol{H}_K\boldsymbol{x}_{K|K-1}) \\ \boldsymbol{P}_{K|K} =& (\boldsymbol{I} - \boldsymbol{K}_K\boldsymbol{H}_K)\boldsymbol{P}_{K|K-1}(\boldsymbol{I} - \boldsymbol{K}_K\boldsymbol{H}_K)^T \\ &+ \boldsymbol{K}_K\boldsymbol{R}_K\boldsymbol{K}_K^T \end{aligned} \tag{5.6}$$

For a description of the matrices see Section 2.1.6. The state covariance $\boldsymbol{P}_{K|K}$ is formulated in the Joseph form to avoid numerical rounding issues. Note that the constant process noise covariance $\boldsymbol{Q}_{K-1}$ and the measurement noise covariance $\boldsymbol{R}_K$ are used as constant matrices. The latter defines the confidence in the recent measurement $\boldsymbol{z}_K$. Since the MSE for each scan-to-model aligned pose is not reliable as a metric of how well the alignment worked, the measurement noise covariance is set to $\boldsymbol{R}_K = 0.01 * \boldsymbol{I}$ for each measurement input. $\boldsymbol{R}_K$ is in the denominator of the Kalman gain computation in Eq. 5.5. Lower values in $\boldsymbol{R}_K$, therefore, lead to a higher Kalman gain and a higher trust in the recent measurement values in contrast to the prediction when updating the current state $\boldsymbol{x}_{K|K}$ (Eq. 5.6).

The scan-to-model aligned poses prior to the candidate ${}^{W}\tilde{\boldsymbol{T}}_{D_{t-M:t-1}}$ are used as measurement input $\boldsymbol{z}_K$ to the EKF. The robot localization package[1] of the ROS framework is used for the EKF implementation. However, by default, the EKF operates in a periodic manner, i.e., with a fixed update rate. If no measurement is received, usually only predictions are performed. The implementation is adjusted to enforce exactly one prediction and one correction step for each measurement input. Since each evaluation sequence is independent and not necessarily connected to the previous sequence, the EKF and its motion model must be reset for each evaluation. Hence, the EKF is re-initialized for each short input sequence of model-aligned poses. This includes the state $\boldsymbol{x}_{K-1|K-1}$, the state transition function $f(\boldsymbol{x}_{K-1|K-1})$, its Jacobian $\boldsymbol{F}_{K-1}$ and also the state covariance $\boldsymbol{P}_{K-1|K-1}$.

The last predicted state $\boldsymbol{x}_{K|K-1}$ after the last measurement input ${}^{W}\tilde{\boldsymbol{T}}_{D_{t-1}}$ is used as a motion prior. The predicted state can be converted to a homogeneous transformation matrix and compared to the last model-aligned pose ${}^{W}\tilde{\boldsymbol{T}}_{D_t}$. As thresholds, the following parameters are defined: If the rotational error is $< 0.5\,\text{deg}$ and the translational error is $< 0.05\,\text{m}$, ${}^{W}\tilde{\boldsymbol{T}}_{D_t}$ is close to the motion prior and can be considered as a good candidate for TMO. In this case, the candidate is passed on to the PGO in the next step. If it is not close to the motion prior, it does not follow the motion model of the previous model-aligned poses. The results

[1] http://wiki.ros.org/robot_localization



(a) Trajectory consisting of map-optimized poses (gray) along one side of an airplane as reference object

(b) Map-optimized poses with indices $4-6$ are corrected with scan-to-model alignments, resulting in model-aligned poses (blue)

(c) Motion prior (red) comparison with the candidate (green)

Figure 5.3: Illustration of the procedure for candidate evaluation with an EKF. Map-optimized poses (gray) are used as an initial guess for the scan-to-model alignment (a). A sequence of map-optimized poses is then refined with individual scan-to-model alignments, resulting in model-aligned poses (blue) (b). The sequence of model-aligned poses is used as input to an EKF (c). If the motion prior of the EKF (red) is close to the last model-aligned pose (green = candidate pose), the attempt is considered successful and the candidate will be used for TMO in the following steps.

of the evaluation can be represented in the form of a confusion matrix:

- **True positive:** The candidate is close to the motion prior and indeed has a very low APE.
- **True negative:** The candidate is not close to the motion prior and has a higher APE.
- **False positive:** The candidate is close to the motion prior but actually has a higher APE.
- **False negative:** The candidate is not close to the motion prior but actually has a very low APE.

The false-positive case occurs, when the candidate pose is close to the motion prior by chance. However, due to the strict thresholds mentioned above, mostly false negatives occur: Some of the model-aligned scans in the sequence might have converged to a wrong pose, causing the motion prior to being off. Since the candidate pose might still have a very low APE, this can be considered a false negative. In any case, if scans did not converge well in the sequence, the slices may be ambiguous and are not suitable for reliable high-accuracy TMO.

Figure 5.3 illustrates the procedure of candidate evaluation. If it was successful, only the last model-aligned pose ${}^{W}\tilde{\boldsymbol{T}}_{D_t}$ (green) is considered for the next steps.

5.2.4 Pose Graph Optimization (PGO)

Once the candidate pose is confirmed, it is assumed that it follows the motion model of the previous model-aligned poses and therefore has a very low rotational and translational error. Hence, it is suitable to be inserted into a pose graph with very high confidence about the absolute pose in the global frame. In the PGO module, all map-optimized poses received from the mapping module of LOAM are inserted into a pose graph structure. A pose graph $\mathcal{G} = \{n_0, n_1, \cdots\}$ consists of graph nodes n. The graph node can be written as a tuple $n_i = \langle \boldsymbol{q}_i, \boldsymbol{t}_i \rangle$ or alternatively, as a transformation matrix of the map-optimized pose $n_i = {}^{W}\boldsymbol{T}_{D_i}$. Edges $\mathcal{E} = \{e_{0,1}, e_{1,2}, \cdots\}$ connect the graph nodes. Each edge $e_{i,j} = \langle \boldsymbol{\Omega}_{i,j}, \boldsymbol{q}_{i,j}, \boldsymbol{t}_{i,j} \rangle$ consists of a 6×6 information matrix $\boldsymbol{\Omega}_{i,j}$ and the relative transform between the two nodes n_i and n_j, which the edge is connecting. The information matrix essentially specifies the confidence in the relative transform. In practice, it can be derived from the inverse of the covariance matrix, e.g., from sensor data. The Ceres Solver implementation [123] is used for the PGO. It is modeled as a nonlinear least-squares optimization problem with the total cost J:

$$\begin{aligned} J(\hat{\boldsymbol{q}}_{i,j}, \hat{\boldsymbol{t}}_{i,j}) &= \sum_{e \in \mathcal{E}} \rho(||f(e_{i,j}, \hat{\boldsymbol{q}}_{i,j}, \hat{\boldsymbol{t}}_{i,j})||_2^2)\,, \\ \text{with} \quad f(e_{i,j}, \hat{\boldsymbol{q}}_{i,j}, \hat{\boldsymbol{t}}_{i,j}) &= \hat{\boldsymbol{\Omega}}_{i,j} \begin{pmatrix} \Delta \boldsymbol{t}_{i,j} \\ 2\Delta \overrightarrow{\boldsymbol{q}}_{i,j} \end{pmatrix}, \\ \text{with} \quad \Delta \boldsymbol{t}_{i,j} &= \boldsymbol{t}_{i,j} - \hat{\boldsymbol{t}}_{i,j} \\ \text{and} \quad \Delta \boldsymbol{q}_{i,j} &= \boldsymbol{q}_{i,j} \hat{\boldsymbol{q}}_{i,j}^{*} \end{aligned} \tag{5.7}$$

$\hat{\boldsymbol{q}}_{i,j}$ and $\hat{\boldsymbol{t}}_{i,j}$ are computed from the iteratively optimized graph nodes n_i and n_j. $\hat{\boldsymbol{\Omega}}_{i,j}$ is the lower triangular matrix of the Cholesky-decomposed information matrix $\boldsymbol{\Omega}_{i,j}$. From the equation, it can be seen, that the information matrix is used as a weight in the cost function formulation. A higher value gives a higher weight to the $\Delta \boldsymbol{t}_{i,j}$ or $\Delta \overrightarrow{\boldsymbol{q}}_{i,j}$ residuals. $f(\cdot)$ is the cost function computing the weighted residual vector and is wrapped inside a Huber loss function $\rho(\cdot)$. $\overrightarrow{\boldsymbol{q}}_{i,j}$ represents the vector part of $\boldsymbol{q}_{i,j}$. The last two terms effectively compute the translational and rotational residuals $\Delta \boldsymbol{t}_{i,j}$ and $\Delta \boldsymbol{q}_{i,j}$, respectively. $\boldsymbol{q}_{i,j}$ and $\boldsymbol{t}_{i,j}$ remain static during the optimization process and are taken from the edge $e_{i,j}$.

A new subgraph $\mathcal{G}_{t-x,t} = \{n_{t-x}, n_{t-x+1}, \cdots, n_t\}$ is created between two successful TMOs at times $t - x$ and t. Each map-optimized pose ${}^{W}\boldsymbol{T}_{D_{t-x+1:t-1}}$ received since then is inserted as a node n_i. Only optimizing the most recent subgraph overcomes the problem of growing computational complexity and memory consumption for PGO. Assuming highly accurate candidate poses, there is also no need to perform global PGO.

When building the subgraph optimization problem, the relative poses (i.e., $\boldsymbol{q}_{i,j}$, $\boldsymbol{t}_{i,j}$) between consecutive nodes have to be computed for the edge $e_{i,j}$. Using homogeneous transformation matrices, a relative transform between two nodes n_i and n_j is formulated as:

$${}^{D_i}\boldsymbol{T}_{D_j} = {}^{W}\boldsymbol{T}_{D_i}^{-1}\,{}^{W}\boldsymbol{T}_{D_j} \tag{5.8}$$

The transformation matrix ${}^{D_i}\boldsymbol{T}_{D_j}$ can then be converted to a quaternion $\boldsymbol{q}_{i,j}$ and a translation vector $\boldsymbol{t}_{i,j}$ representation to form the edge $e_{i,j}$. All edges of map-optimized poses are inserted with the information matrix $\boldsymbol{\Omega}_{i,j} = \boldsymbol{I}$, giving them equal weight in the subgraph.

Once a candidate pose is confirmed from the previous step, the scan-to-model aligned pose ${}^{W}\tilde{\boldsymbol{T}}_{D_t}$ is inserted with a very high confidence, e.g., $\boldsymbol{\Omega}_{t-x,t} = 4000 \cdot \boldsymbol{I}$ relatively to the previous candidate pose ${}^{W}\tilde{\boldsymbol{T}}_{D_{t-x}}$. Hence, only the nodes and edges of the current subgraph need to be kept and all prior to $t - x$ can be deleted to overcome the growing graph problem. The relative transform between the previous and current candidate poses can be computed with Eq. 5.8: ${}^{D_{t-x}}\tilde{\boldsymbol{T}}_{D_t} = {}^{W}\tilde{\boldsymbol{T}}^{-1}_{D_{t-x}} {}^{W}\tilde{\boldsymbol{T}}_{D_t}$.

Finally, the PGO problem in Eq. 5.7 is solved with the Levenberg-Marquardt trust-region algorithm [77], [78]. The first node n_{t-x} and last node n_t in the subgraph are set as constant, which are the previous and current candidate poses, respectively. This ensures that the solver can only adjust the map-optimized poses ${}^{W}\boldsymbol{T}_{D_{t-x+1:t-1}}$ according to the constraint ${}^{D_{t-x}}\tilde{\boldsymbol{T}}_{D_t}$. The ultimate output of the PGO module are the pose graph-optimized (PG-optimized) poses ${}^{W}\bar{\boldsymbol{T}}_{D_{t-x:t}}$.

5.2.5 Map correction

The task of the map correction module is to correct the map of the LOAM algorithm with the PG-optimized poses ${}^{W}\bar{\boldsymbol{T}}_{D_{t-x:t}}$ from the previous step. As described previously, LOAM first computes the map-optimized transform, before inserting the LiDAR scan into the map with this transform. LOAM voxelizes the map after each scan insertion, effectively merging the new scan points into the voxels. Hence, a correction in the sense of individually correcting the already inserted scan points with their respective PG-optimized poses is not possible. Instead, a map rebuilding algorithm is proposed. It takes the map snapshot of the previous TMO at time $t - x$ and re-inserts buffered LiDAR scans with their new PG-optimized poses until the most recent TMO at time t.

Algorithm 4 shows the pseudo-code of the map correction module. As input serve on one hand the buffered corner and surface features of each LiDAR scan $\mathcal{P}_{\mathcal{E}}^{D_{t-x:t+n}}$ and $\mathcal{P}_{\mathcal{H}}^{D_{t-x:t+n}}$, respectively. On the other hand, the PG-optimized poses ${}^{W}\bar{\boldsymbol{T}}_{D_{t-x:t}}$ are required. The proposed reference object-based TMO is a parallelized and possibly offloaded process on an Edge Cloud. Hence, the LOAM algorithm on the robotic platform continues processing LiDAR scans while the proposed extension attempts to find a good candidate pose for TMO. In other words, one TMO attempt may take several seconds on the Edge Cloud, during which time the conventional LOAM algorithm continues pose estimation and map insertion. The computed map-optimized poses after the TMO candidate are denoted as ${}^{W}\boldsymbol{T}_{D_{t+1:t+n}}$ and also serve as input to the algorithm.

It first creates new feature maps and, if existing, inserts the feature map snapshots taken in the previous map rebuilding iteration (lines 4+5). The scan features captured at a time t_i are first downsampled (line 8), transformed to the world frame with the PG-optimized pose (line 9), and then inserted into the new feature maps (lines 11+12). The parameters $e = 0.2\,\mathrm{m}$ and $h = 0.4\,\mathrm{m}$ are resolution parameters for the voxelization. It was empirically determined that it is best to voxelize the new feature maps after each scan insertion instead of once after all insertions (line 13). This increases the computational load but has shown the best performance by means of APE and RE. Snapshots of the new feature maps are taken, once all scans between the time $t - x$ and t are inserted (lines 15+16), which are then used as base maps for

Algorithm 4: Pseudo-code for map rebuilding adapted to the LOAM algorithm [12]

```
   Input: $\mathcal{P}_{\mathcal{E}}^{D_{t-x:t+n}}, \mathcal{P}_{\mathcal{H}}^{D_{t-x:t+n}}, {}^{W}\bar{\boldsymbol{T}}_{D_{t-x:t}}, {}^{W}\boldsymbol{T}_{D_{t+1:t+n}}$
   Output: $\overrightarrow{\mathcal{P}}_{\mathcal{E}}^{W}, \overrightarrow{\mathcal{P}}_{\mathcal{H}}^{W}$
1  foreach TMO at time $t$ do
2      if not the first TMO then
3          /* Use map snapshot from previous TMO as a base for the new feature maps */
4          $\overrightarrow{\mathcal{P}}_{\mathcal{E}}^{W} \leftarrow {}^{\text{snap}}\overrightarrow{\mathcal{P}}_{\mathcal{E}}^{W}$
5          $\overrightarrow{\mathcal{P}}_{\mathcal{H}}^{W} \leftarrow {}^{\text{snap}}\overrightarrow{\mathcal{P}}_{\mathcal{H}}^{W}$
6      /* Insert scan features with the PG-optimized poses */
7      foreach $t_i$ in $t - x : t$ do
8          $\mathcal{P}_{\mathcal{E}}^{D_{t_i}}, \mathcal{P}_{\mathcal{H}}^{D_{t_i}} \leftarrow$ downsample($\mathcal{P}_{\mathcal{E}}^{D_{t_i}}, \mathcal{P}_{\mathcal{H}}^{D_{t_i}}, e, h$)
9          $\mathcal{P}_{\mathcal{E}}^{W_{t_i}}, \mathcal{P}_{\mathcal{H}}^{W_{t_i}} \leftarrow$ transform( $\mathcal{P}_{\mathcal{E}}^{D_{t_i}}, \mathcal{P}_{\mathcal{H}}^{D_{t_i}}, {}^{W}\bar{\boldsymbol{T}}_{D_{t_i}}$)
10         /* Add scan to new feature maps */
11         $\overrightarrow{\mathcal{P}}_{\mathcal{E}}^{W} \leftarrow \overrightarrow{\mathcal{P}}_{\mathcal{E}}^{W} + \mathcal{P}_{\mathcal{E}}^{W_{t_i}}$
12         $\overrightarrow{\mathcal{P}}_{\mathcal{H}}^{W} \leftarrow \overrightarrow{\mathcal{P}}_{\mathcal{H}}^{W} + \mathcal{P}_{\mathcal{H}}^{W_{t_i}}$
13         $\overrightarrow{\mathcal{P}}_{\mathcal{E}}^{W}, \overrightarrow{\mathcal{P}}_{\mathcal{H}}^{W} \leftarrow$ downsample($\overrightarrow{\mathcal{P}}_{\mathcal{E}}^{W}, \overrightarrow{\mathcal{P}}_{\mathcal{H}}^{W}, e, h$)
14     /* Take snapshot of new feature maps */
15     ${}^{\text{snap}}\overrightarrow{\mathcal{P}}_{\mathcal{E}}^{W} \leftarrow \overrightarrow{\mathcal{P}}_{\mathcal{E}}^{W}$
16     ${}^{\text{snap}}\overrightarrow{\mathcal{P}}_{\mathcal{H}}^{W} \leftarrow \overrightarrow{\mathcal{P}}_{\mathcal{H}}^{W}$
17     ${}^{W}\hat{\boldsymbol{T}}_{D_{t+1:t+n}} \leftarrow$ rebase(${}^{W}\boldsymbol{T}_{D_{t+1:t+n}}, {}^{W}\bar{\boldsymbol{T}}_{D_t}$)
18     /* Repeat (7-13) for $t + 1 : t + n$ with ${}^{W}\hat{\boldsymbol{T}}_{D_{t_i}} \in {}^{W}\hat{\boldsymbol{T}}_{D_{t+1:t+n}}$ */
19     /* Request mapping lock */
20     /* Update new map in the mapping module */
21     updateMap($\overrightarrow{\mathcal{P}}_{\mathcal{E}}^{W}, \overrightarrow{\mathcal{P}}_{\mathcal{H}}^{W}$)
22     /* Adjust current pose in mapping module */
23     updateCurrentPose(${}^{W}\hat{\boldsymbol{T}}_{D_{t+n}}$)
24     /* Request mapping unlock */
```

the next map rebuilding iteration. It is important to note that the scan mapping process in the mapping module continues even while the new map is being created. In this notation, we assume that since the TMO pose at time t, several scans were processed in the mapping module until time $t + n$, producing the map-optimized transforms ${}^{W}\boldsymbol{T}_{D_{t+1:t+n}}$. These poses are also called the *tail* of the current subgraph. Since these poses are still based on the old map-optimized pose ${}^{W}\boldsymbol{T}_{D_t}$, they need to be adapted to the TMO pose ${}^{W}\bar{\boldsymbol{T}}_{D_t}$ (line 17) yielding the new tail poses ${}^{W}\hat{\boldsymbol{T}}_{D_{t+1:t+n}}$. The scans of the tail are then inserted with these poses into the new map (line 18). The poses of the tail have not been PG-optimized yet and are just inserted to keep up with the online mapping process. For this reason, the snapshots of the new feature maps were created previously, which only contain scans inserted with PG-optimized poses. Once the new map is fully built, the mapping module is locked, its feature maps are replaced (line 21), and the latest robot pose ${}^{W}\boldsymbol{T}_{D_{t+N}}$ is updated to ${}^{W}\hat{\boldsymbol{T}}_{D_{t+N}}$ (line 23). While the lock, no LiDAR scans are processed in the mapping module. However, the odometry module of the LOAM framework still provides pose estimates, see Section 2.2.1 for more details. If implemented efficiently, only memory pointers need to be exchanged when replacing the maps. This will lock the mapping process for only a few milliseconds, which is negligible assuming a LiDAR rate of 10 Hz. The mapping module is then unlocked and it continues map-optimization of new scans with the new feature maps.

The map correction module is the only module in the pipeline shown in Fig. 5.1, which is tightly coupled to the LiDAR-SLAM algorithm. This is due to the unique way maps are built in each algorithm. Therefore, the map correction module needs to be adapted correspondingly.

5.3 Experimental setup

5.3.1 Ground-truth (GT)

One of the biggest challenges for method evaluation is the generation of GT data in real-world scenarios. The proposed method utilizes the known pose and geometry of a reference object to improve LOAM accuracy. Hence, three requirements must be fulfilled to establish a reliable GT:

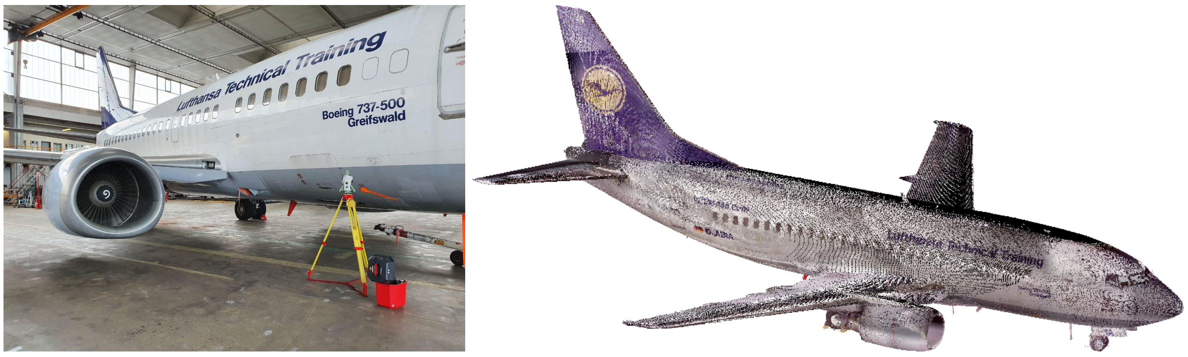
1. The 3D model of the reference object must be available, preferably with nearly perfect geometry.
2. The pose of the reference object must be known in a global or map coordinate frame.
3. The GT system should be mobile, easy to set up and generate highly accurate measurements at a high update rate.

The **first** requirement expects a highly accurate point cloud or triangular mesh of the reference object. Unless the manufacturer of the reference object provides a 3D CAD model, a triangular mesh is not straightforward to obtain. Commonly, a stationary 3D scanning device, e.g., from BIM is used to generate a dense point cloud of the environment or reference object. It can then be converted to a triangular mesh, see Section 2.1.3.2. This is, typically, highly dependent on the correct estimation of point normals.

The **second** requirement can be achieved by performing scan-to-model alignments with the LiDAR sensor prior to the exploration when the robotic platform is still static. If single scans are insufficient, an accumulated scan with the actuator transforms can be used.

The **third** requirement demands a GT system, which can be easily transported and moved to another location. At the same time, it should be easy to set up. Often, an infrared camera system is used to establish GT in robotic applications, e.g., "Vicon". However, such a system is fixed to a specific room, has limited area coverage, is expensive, and requires an extrinsic calibration of the cameras.

To evaluate the proposed approach in a large-scale indoor environment with time-limited access (e.g., in an airplane maintenance hangar), a prism tracking system was found to be the



(a) Ground-truth system during 3D model generation (b) Generated highly accurate 3D model of a B737 airplane

Figure 5.4: Illustration of the Leica GT system during 3D model generation (a). With its selective scanning functionality, a highly accurate 3D model of the airplane as a reference object could be generated (b).

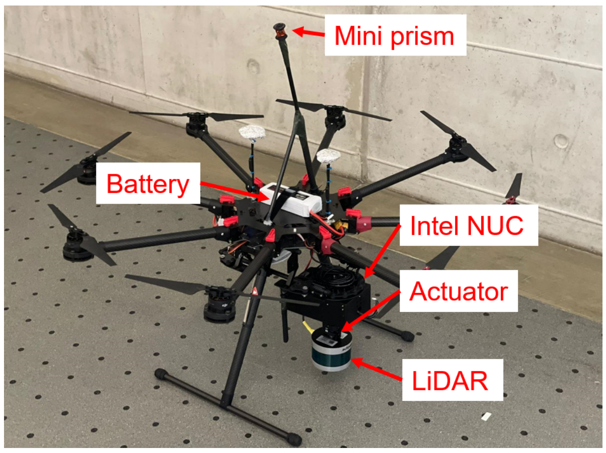
most suitable solution. Specifically, a Leica Nova MS60 MultiStation is able to track the 3-DoF position of a prism with an update rate of up to 20 Hz. It is easy to set up within a few minutes at arbitrary locations inside an environment with extrinsically calibrated landmarks. When initializing the Leica coordinate system, the landmarks (e.g., Leica tapes or surface landmarks) are measured and calibrated.

5.3.2 3D model generation

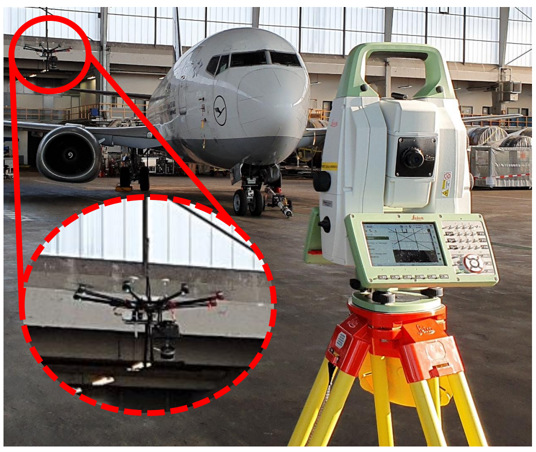
Besides recording the GT position of the prism on the UAV, the system can also create point clouds with selective scanning. With this functionality, a highly accurate 3D model of an airplane as a reference object was created. Figure 5.4(a) shows the Leica system during the 3D model generation. In (b), the point cloud of the final 3D model can be seen. The Leica system had to be re-positioned several times to acquire a complete representation. It uses a triangulation method with at least three calibrated markers in the environment to relocalize itself after a position change. Repositioning can be performed with submillimeter accuracy. Since all acquired scans are in the same Leica coordinate frame, no further point cloud merging or registration is required. However, manual isolation of the points is necessary, since some parts of the environment are also included in the selective scanning. The 3D model consists of $2.9M$ points with an average nearest neighbor distance of $0.88\,\mathrm{cm}$. Due to varying sampling densities, the model is downsampled with a $1\,\mathrm{cm}$ voxel filter. The resulting point cloud has $1.6M$ points and an average nearest neighbor distance of $1.4\,\mathrm{cm}$. This point cloud of the B737 airplane is used as a reference model for the following experiments.

5.3.3 Dataset recording

As a robotic platform, an octocopter UAV is equipped with a Velodyne VLP-16 LiDAR sensor mounted on a gimbal. It is continuously actuated back and forth between $\pm 40°$ around the roll axis. Figure 5.5(a) shows the octocopter UAV with its components and (b) shows the

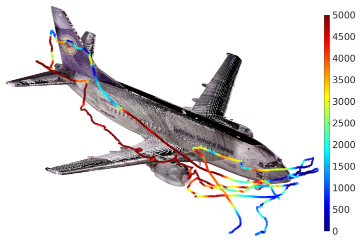


(a) Octocopter UAV

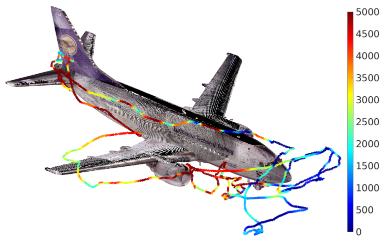


(b) Leica GT system

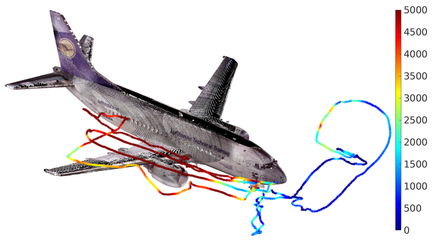
Figure 5.5: Illustration of the UAV platform (a) and the Leica GT system during dataset recording (b). The octocopter UAV is equipped with an actuated VLP-16 LiDAR and a low-weight Leica 360° mini prism. The highly accurate 3-DoF position of the prism is tracked with a Leica MS60 MultiStation displayed in (b). Adapted from [2], ©2022 IEEE.



(a) Dataset 1



(b) Dataset 2



(c) Dataset 3

Figure 5.6: Illustration of the trajectories for each dataset along a B737 airplane used as a reference object. The colors indicate the number of scan points on the reference object captured with the LiDAR sensor at the corresponding position. Adapted from [2], ©2022 IEEE.

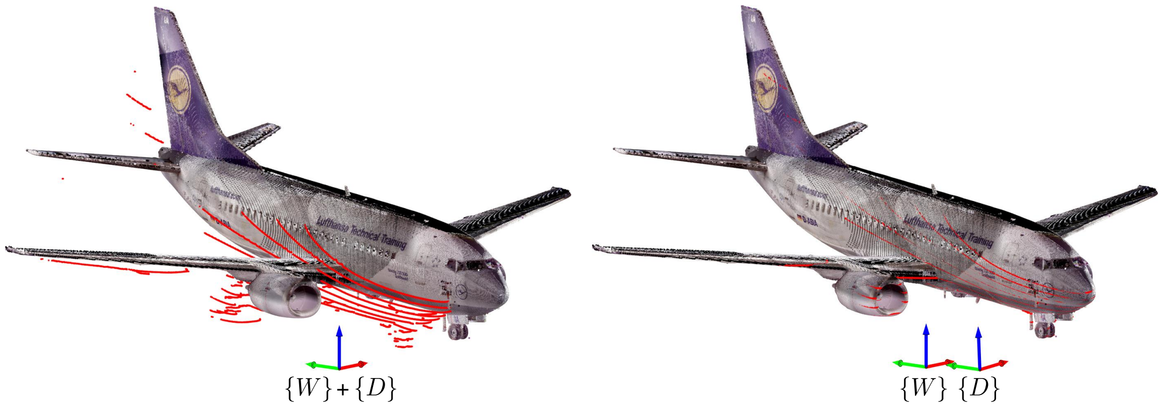
Leica GT tracking system during a dataset recording. An Intel NUC stores captured LiDAR data on an SSD drive. A low-weight Leica 360° mini prism is mounted on a rod on top of the UAV. The high mount ensures permanent direct sight to the tracking system even during the flight. The UAV is manually controlled by an operator.

Three datasets were recorded in a visual airplane inspection scenario. Figure 5.6 shows the trajectories of the generated datasets on one side of the B737 airplane used as a reference object. Depending on the position of the UAV, and the LiDAR rotation, the airplane is not visible at all times in the LiDAR scans. To illustrate this, the number of points belonging to the reference object is color-coded and mapped onto the trajectories. Dark blue color indicates few points and dark red color many points, respectively. It can be seen that the reference object is mostly visible from the side. At the front of the airplane, few to no points remain after the scan isolation.

Table 5.1 shows the metadata of each dataset. The recording time ranges from 8 minutes to over 11 minutes, limited by the battery capacity of the UAV. The average velocity amounts

Table 5.1: Metadata for the three generated datasets.

dataset	#scans	avg. vel.	length	duration
1	5076	$0.33\,\mathrm{m/s}$	$162\,\mathrm{m}$	8m 32s
2	7034	$0.30\,\mathrm{m/s}$	$211\,\mathrm{m}$	11m 49s
3	5712	$0.39\,\mathrm{m/s}$	$196\,\mathrm{m}$	9m 36s



(a) First 20 accumulated LiDAR scans (red) before the airplane pose estimation

(b) LiDAR scans (red) after the pose estimation with scan-to-model alignment

Figure 5.7: Illustration of the reference object pose estimation process. In (a), the first 20 accumulated LiDAR scans at the starting position of the UAV can be seen. The point cloud of the airplane (colored) is in the world coordinate frame $\{W\}$ (= origin) and the LiDAR scans (red) are in the robot frame $\{D\}$, which is also at the origin initially. To estimate the real relative pose between the LiDAR on the UAV with frame $\{D\}$ and the airplane model in frame $\{W\}$, a scan-to-model alignment is performed. The result can be seen in (b). The LiDAR scans converged towards the model surface. After the convergence, the relative pose of $\{D\}$ in $\{W\}$ is known.

to $0.3\,\mathrm{m/s}$ to $0.4\,\mathrm{m/s}$. The major differences between the datasets are the inspection of the vertical stabilizer in Dataset 1 (Fig. 5.6(a)), the inspection of the right wing of the airplane in Dataset 2 (Fig. 5.6(b)), and the partial exploration of the other side of the airplane in Dataset 3 (Fig. 5.6(c)).

5.3.4 Trajectory alignment

In order to evaluate the error of the SLAM trajectory, the estimated poses need to be transformed to the same coordinate frame as the GT. Methods for trajectory alignment were described in Section 2.1.5.1. Umeyama alignment, for example, registers the poses of the SLAM algorithm to the GT by estimating a transform. However, this might remove drift. Origin alignment first computes the relative transform of the first timestamp-aligned 6-DoF poses between the SLAM and GT trajectory. This transform is then applied to all poses. The Leica system, however, only provides the 3-DoF position and no orientation information. Hence, origin alignment can not be applied.

Instead, it is leveraged that the 3D model's point cloud is already recorded in the GT coordinate system. By estimating the initial relative pose between the LiDAR and the airplane model, the resulting transform can also be used to align the SLAM trajectory to the GT. However, this can only be leveraged because the airplane point cloud is in the Leica/GT coordinate system.

To estimate the relative pose to the reference object, the following procedure is followed: The first LiDAR scans captured at the starting position of the UAV can be aligned to the 3D model with the scan-to-model formulation described in Eq. 5.2. Figure 5.7 illustrates the process of relative pose estimation using scan-to-model alignment. By default, all points of the reference model are in a coordinate system at the origin, e.g., world $\{W\}$ in this example (see Fig. 5.7(a)). Captured LiDAR scans in the robot frame $\{D\}$ are also at the origin by default, hence $\{W\} \equiv \{D\}$. However, the robot may not be positioned exactly at the origin of the reference model as can be seen in the offset between the scans and the airplane point cloud. By performing scan-to-model alignment with a suitable initial guess, the relative transform between the UAV and the airplane can be estimated. After the alignment, this transform is then essentially $\{W\} \rightarrow \{D\}$. With the transform ${}^{W}\boldsymbol{T}_{D}$ and the model point cloud, R-LOAM (Chapter 4) and the proposed method (RO-LOAM) are initialized and operational. One should note, that the LiDAR scan in Fig. 5.7(a) is already relatively close to the reference point cloud due to a good initial guess. If it is completely off, the initial guess before the scan-to-model alignment needs to be adjusted manually or a global registration method is required beforehand to yield a coarse alignment.

The resulting relative transform ${}^{W}\boldsymbol{T}_{D}$ is used in the following experiments to align the GT and SLAM trajectories.

5.3.5 Timestamp synchronization

In Section 2.1.5.2, the APE and RE evaluation metrics were explained and used in the previous chapters with simulated data. These metrics assume that for each LiDAR scan, a GT transform/pose exists, i.e., the LiDAR scan and GT pose have the same timestamps. In practice, however, the clocks of the GT system and the robot may not be synchronized or run at a different frame rate. Hence, exactly identical timestamps do not exist.

To find the timestamp offset between the Leica GT system and each of the datasets, a sliding window approach based on the MSE between the GT and SLAM trajectories is followed: R-LOAM in offline mode with 35 iterations is assumed to have the best overall performance, and is used to compute the SLAM trajectory. Then, the relative transform ${}^{W}\boldsymbol{T}_{D}$ computed in the previous step is applied to transform the R-LOAM trajectory to the world/Leica GT frame. The Leica and R-LOAM trajectories are now in the same coordinate frame but still have mismatching timestamps. Figure 5.8 shows the Z-profiles of R-LOAM and the GT using Dataset 1. The profiles are coarsely aligned, but the timeline is still a bit shifted.

Now, an empirically determined time offset is added to either of the trajectories and the nearest neighbors in the time domain are found for all poses. Subsequently, the MSE is computed for all poses. The time offset is then adjusted within the sliding window range, and the process is repeated. Hence, for each time offset, an MSE value is retrieved. Figure 5.9(a)

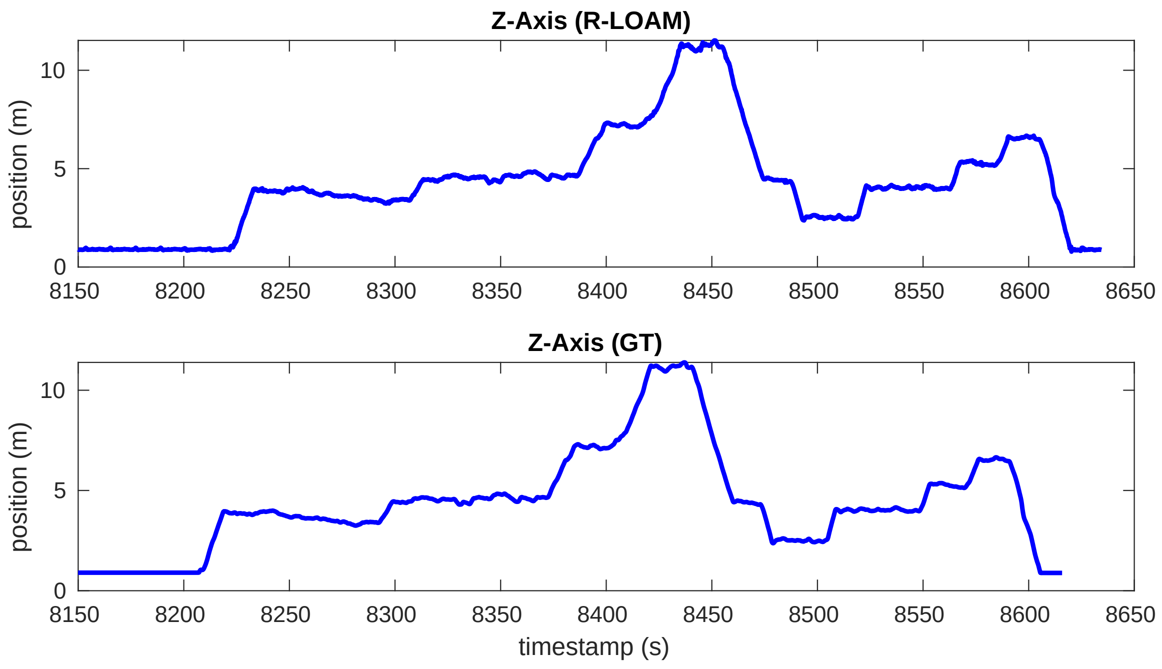
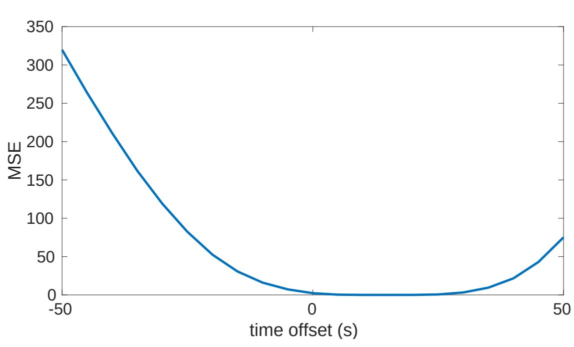
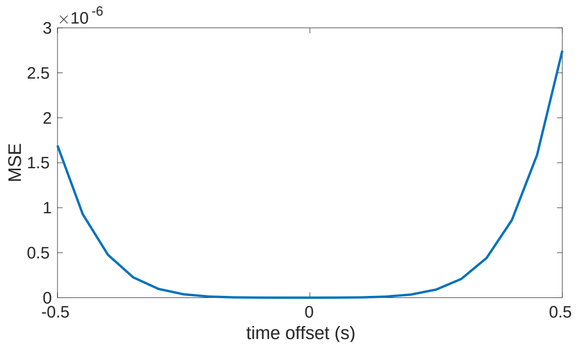


Figure 5.8: Z-profiles of the coarsely timestamp-aligned R-LOAM and GT trajectories using Dataset 1. Still, a time offset can be seen when comparing the profiles, which needs to be determined and corrected before the APE and RE computation.



(a) MSE curve of a time window for the coarsely timestamp-aligned profiles displayed in Fig. 5.8 using all axes



(b) Final MSE curve after the fine timestamp synchronization and adjusting the time offset

Figure 5.9: Illustration of the MSE curves for coarse (a) and fine (b) timestamp synchronization. By narrowing down the time window and step size, a highly accurate timestamp synchronization can be achieved. The step size in (b) is $50\,\mathrm{ms}$ and the global minimum is at the time offset 0. This means that the R-LOAM and GT trajectories have fully synchronized timestamps.

shows the MSE curve of a coarse timestamp synchronization using all axes. The current time offset is still not at the global minimum and hence, still needs to be further adjusted. After reducing the time window and step size, the global minimum is found with an accuracy of $50\,\mathrm{ms}$ (Fig. 5.9(b)). The VLP-16 LiDAR sensor has an update rate of $10\,\mathrm{Hz}$, which amounts to an incoming scan every $100\,\mathrm{ms}$. Hence, the accuracy in timestamp synchronization of $50\,\mathrm{ms}$ is already below the update rate of the LiDAR sensor and can be considered optimal for the respective dataset.

Another intuitive explanation for the timestamp synchronization is as follows: The GT profile in Fig. 5.8 is shifted along the timeline and each time the MSE between the SLAM and GT trajectories is computed. This essentially generates the curve as displayed in Fig. 5.9.

The time offset retrieved with this sliding window approach can be applied to all experiments of the same dataset. Due to varying time offsets, the process of timestamp synchronization may need to be repeated for each dataset.

5.4 Experimental results

With the correct transformation between the SLAM and GT system and the correct time synchronization, the trajectory error can be evaluated by means of APE. Since there is still no exact matching timestamp of a GT pose to a LiDAR scan, the GT pose is linearly interpolated for the corresponding LiDAR scan to achieve an even higher quality comparison.

For the following results, the experiments are performed in *online* mode. The experiments from the previous chapters were presented in *offline* mode for reproducibility. However, the proposed method in this chapter is supposed to run in a fully parallelized manner on an Edge Cloud, and hence, the operation in *online* mode is desirable. Here, LiDAR scans are inserted into the pipeline with $10\,\mathrm{Hz}$ corresponding to the frequency of the LiDAR sensor. To cope with the amount of data and to avoid buffer overflow, frames are dropped in the mapping module of LOAM and only the latest scan features in the buffer are processed. The results are not fully reproducible in online mode due to the parallelized manner and random frame drops. Hence, each parameter configuration is repeated five times and the average results are presented. All experiments in this section were conducted on a server with 32 Intel Xeon CPUs E5-2690 @ 2.90 GHz and 132 GB memory. The server can be considered suitable as an Edge Cloud.

Also, the convergence time in the scan-to-model alignment module is limited to $2000\,\mathrm{ms}$. This avoids that time-consuming, bad convergences block the pipeline for too long. Instead, the scan-to-model aligned poses converged until this time limit are evaluated by the candidate evaluation module. Wrongly converged poses will not pass the motion prior test and the next scan sequence is processed. Usually, a well-fitting scan on the reference object with low residuals converges quickly and does not reach this time limit.

5.4.1 Parameter determination

The proposed pipeline has a variety of parameters, which are listed below:

- **#points:** Scan downsampling parameter. The points remaining after scan isolation are uniformly downsampled to this number before the scan-to-model alignment.

- **#iter:** Number of iterations for scan-to-model alignment. A higher number may result in better converged and refined poses but requires more time. A lower number may be sufficient depending on the accuracy of the map-optimized pose used as an initial guess.

#points	APE mapping (cm)				#TMOs
	max	mean	median	RMSE	
500	84.2	**8.9**	**6.5**	**13.6**	**62**
1000	80.3	10.4	7.2	15.8	49
5000	78.2	12.6	8.6	18.7	38
10000	**75.4**	11.7	8.0	17.7	41

Table 5.2: APE for varying #points using Dataset 1. After scan isolation and before scan-to-model alignment, the scans are downsampled to #points. The other parameters are set to $L = 15$, $M = 9$, #iter $= 100$. The map-optimized poses of LOAM are used as initial guesses. The best results are marked in **bold**.

#iter	APE mapping (cm)				#TMOs
	max	mean	median	RMSE	
50	167	39.0	25.6	55.1	50
100	84.2	**8.9**	**6.5**	**13.6**	**62**
200	**81.2**	10.3	7.1	15.6	48

Table 5.3: APE for varying #iter using Dataset 1. The parameter controls the number of scan-to-model alignment iterations. The other parameters are set to $L = 15$, $M = 9$, #points $= 500$. The map-optimized poses of LOAM are used as initial guesses. The best results are marked in **bold**.

- **M:** Sequence length of map-optimized poses for scan-to-model alignment and candidate evaluation. M map-optimized poses previous to the candidate pose for TMO are refined with scan-to-model alignment. Hence, $M + 1$ map-optimized poses are used for scan-to-model alignment.
- **L:** Buffer size of map-optimized poses from LOAM. After it has been filled, a TMO attempt of the proposed extension is triggered. Lower values result in more attempts and higher values in fewer attempts.

The first experiments investigate suitable values for these parameters.

#points First, it is investigated, if it is harmful to reduce the number of isolated points on the reference object's surface. For this, a uniform downsampling operation is applied to each of the isolated scans in the sequence with length $M + 1$. The results for #points $= \{500, 1000, 5000, 10000\}$ using Dataset 1 are presented in Table 5.2. The other parameters are set to $L = 15$, $M = 9$, #iter $= 100$. It should be noted that the scan-to-model convergence time is limited to $2000\,\mathrm{ms}$ as described above. Hence, isolated scans with many points on the surface may not reach the 100 iterations. The results show that 500 points are sufficient for the visual airplane inspection scenario. Map-optimized poses can be refined with scan-to-model alignments to a very low APE. Also, more TMOs are successful compared to higher #points. This is also the reason for the lower median APE when using #points $= 500$. For the following experiments, #points is set to 500.

#iter The next experiment investigates the influence of #iter on the APE and #TMOs using Dataset 1. Table 5.3 shows the average results of five runs for #iter $= \{50, 100, 200\}$. The other parameters are set to $L = 15$, $M = 9$, #points $= 500$. 50 iterations show the worst performance. One of the five runs failed to correct drift on time and therefore resulted in the same performance as conventional LOAM. The reason is that 50 iterations may not always result in a fully model-converged pose, failing the candidate evaluation step. Once the drift

M	APE mapping (cm)				#TMOs
	max	mean	median	RMSE	
4	79.5	11.7	9.0	16.8	18
9	94.9	**9.9**	**6.3**	16.6	**25**
19	96.7	13.7	9.1	20.4	19
29	**58.5**	11.5	8.9	**15.7**	20

Table 5.4: Experimental results for variations of the parameter M on Dataset 1. $M + 1$ scans are used for the scan-to-model alignment and candidate evaluation. The other parameters are set to $L = 50$, #points $= 500$, #iter $= 100$. The map-optimized poses of LOAM are used as initial guesses. The best results are marked in **bold**. Adapted from [2], ©2022 IEEE.

L	APE mapping (cm)				#TMOs
	max	mean	median	RMSE	
15	**84.2**	**8.9**	**6.5**	**13.7**	**62**
100	96.3	17.0	11.6	24.4	12
200	91.9	17.3	13.3	23.0	6
300	92.6	24.4	17.4	31.0	4

Table 5.5: Experimental results for variations of the parameter L on Dataset 1. A scan-to-model alignment is triggered every L scans. The other parameters are set to $M = 9$, #points $= 500$, #iter $= 100$. The map-optimized poses of LOAM are used as initial guesses. The best results are marked in **bold**. Adapted from [2], ©2022 IEEE.

is too large, the ICP-based scan-to-model alignment can not converge anymore. 100 iterations show the lowest mean and median APE with the highest #TMOs. 200 iterations do not further improve scan-to-model alignment, but instead, take more time to converge. This results in fewer #TMOs. For all following experiments, #iter is set to 100.

M - Sequence length The length of the scan-to-model aligned sequence determines the quality of the motion model in the EKF during the candidate evaluation step. The parameter M controls the length of the sequence. Table 5.4 shows the results for $M = \{4, 9, 19, 29\}$ using Dataset 1. The other parameters are set to $L = 50$, #points $= 500$, #iter $= 100$. L is set to 50 because the length of the sequence M can not be longer than the number of buffered map-optimized poses L. The results show that a short sequence with $M = 4$ is sufficient to establish the motion model and to achieve 18 TMOs. Longer sequences with $M \geq 19$ do neither improve the APE nor yield more TMOs. The reason is that there is a higher risk for scans in the sequence converging to wrong poses and hence fail the candidate evaluation. Also, a longer sequence requires more CPUs. Essentially, #CPUs $= M + 1$ are required for fully parallelized scan-to-model alignment. To summarize, $M = 9$ results in the lowest median APE and the highest #TMOs. This sequence length is applied to the following experiments.

L - Buffer size The proposed pipeline buffers map-optimized poses from the mapping module of LOAM and the corresponding isolated scans. Once L poses and scans have been buffered, a TMO attempt is triggered. This effectively controls the maximum frequency of TMOs. Table 5.5 shows the results for $L = \{15, 100, 200, 300\}$ using Dataset 1 and the map-optimized poses of LOAM as initial guesses. The other parameters are set to $M = 9$, #points $= 500$, #iter $= 100$. It can be seen that a higher value for L significantly reduces the frequency of TMOs and also increases the median and mean APE as well as RMSE. On the other hand, even $L = 300$ with four TMOs results in only $17\,\mathrm{cm}$ median APE compared to $71\,\mathrm{cm}$ of conventional LOAM. The results for LOAM and R-LOAM without the proposed TMO extension

APE	scan-to-model aligned poses (%)		
	Dataset 1	Dataset 2	Dataset 3
$< 10\,\mathrm{cm}$	9	16	7
$< 50\,\mathrm{cm}$	32	41	25
$< 100\,\mathrm{cm}$	50	59	39

Table 5.6: Percentage of scan-to-model aligned poses below a certain APE. The map-optimized poses of the LOAM algorithm [12] were used as initial guesses. Adapted from [2], ©2022 IEEE.

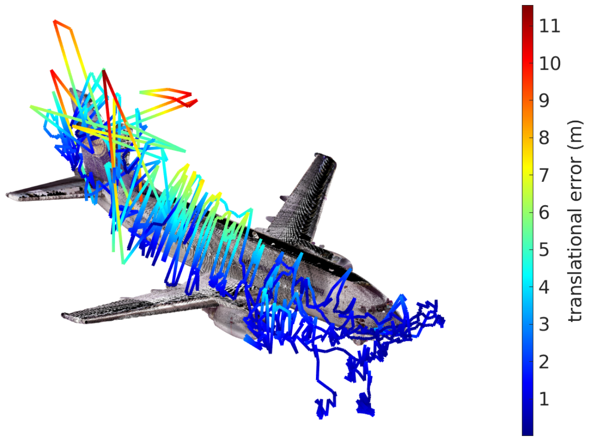


Figure 5.10: Model-aligned poses colored according to their APE(m) using the map-optimized poses of the LOAM algorithm [12] of Dataset 1 as initial guesses. The task of the candidate evaluation module is to find the poses, which have a low translational error.

are presented later on. The best results can be seen for $L = 15$ with only $7\,\mathrm{cm}$ median APE and over 60 successful TMOs. For the following experiments, L is set to 15.

5.4.2 Localization-only approach

As Fig. 5.6 showed, the airplane is not visible at all times in the isolated LiDAR scans. Even if it is visible, alignment ambiguities may not result in a low APE despite a low final MSE. Hence, a localization-only approach leveraging the reference object without a SLAM algorithm is not possible. This is demonstrated by the results in Table 5.6 for the three datasets. It shows the percentage of scan-to-model aligned poses below certain APE thresholds. Dataset 3 is the most challenging with less than 40% model-aligned poses with APE $< 100\,\mathrm{cm}$. A large part of the trajectory was in the front of the airplane with limited reference object visibility (see Fig. 5.6(c)). Scan-to-model aligned poses with APE $< 10\,\mathrm{cm}$ are considered suitable for TMO. Less than 10% of the scan-to-model aligned poses fulfill this criterion. To find out, if a TMO candidate pose is in this range, the EKF-based evaluation step is performed. Ideally, a candidate passing this step has an APE $< 10\,\mathrm{cm}$.

Figure 5.10 illustrates all scan-to-model aligned poses for Dataset 1. The trajectory is color-coded according to the APE. The map-optimized poses of the LOAM algorithm were used as initial guesses. It should be noted that no TMO was actually performed. The jumps in the trajectory are due to alignment ambiguities or failed convergences.

5.4.3 Benchmark against LOAM and R-LOAM

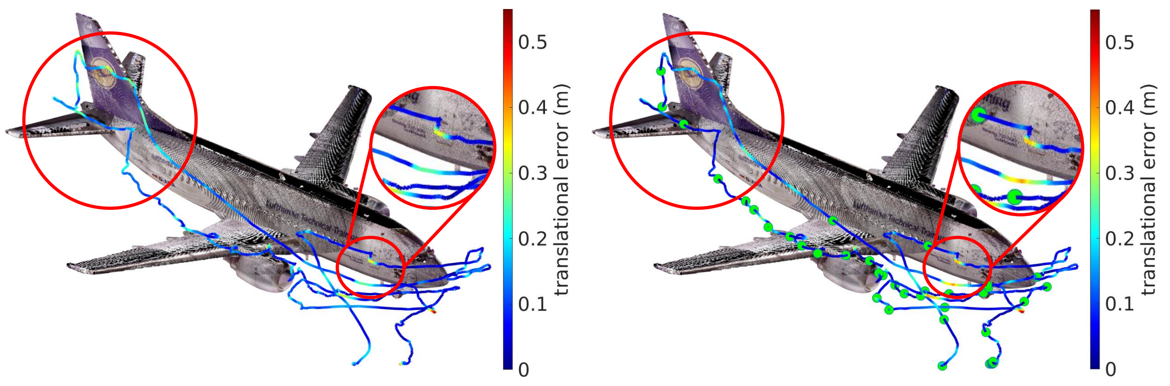
The proposed extension is enabled for the LOAM and R-LOAM algorithms, denoted as LOAM + RO and R-LOAM + RO, respectively. For R-LOAM, the same 3D model as for the proposed TMO extension is leveraged. The results for all three datasets are presented in

Table 5.7: Experimental results for the three datasets. The best results are marked in **bold**. Adapted from [2], ©2022 IEEE.

	method	**APE mapping** (cm)				**freq.** (Hz)	**APE TMO** (cm)				**#TMOs**
		max	mean	median	RMSE	mapping	max	mean	median	RMSE	
Dataset 1	LOAM	466.3	134.6	71.1	182.7	**3.3**	-	-	-	-	-
	LOAM + RO	**84.2**	**8.9**	**6.5**	**13.6**	3.0	20.3	5.2	4.2	6.4	62
	R-LOAM	**50.7**	12.6	11.5	14.4	**3.1**	-	-	-	-	-
	R-LOAM + RO	57.8	**8.1**	**5.9**	**10.5**	**3.1**	22.9	5.2	4.1	6.7	62
Dataset 2	LOAM	423.2	117.1	67.2	164.7	**2.9**	-	-	-	-	-
	LOAM + RO	**89.7**	**9.6**	**6.4**	**15.7**	2.6	18.2	4.0	3.3	5.2	62
	R-LOAM	**58.4**	10.3	9.6	12.4	**2.8**	-	-	-	-	-
	R-LOAM + RO	65.0	**7.2**	**5.2**	**10.4**	2.6	16.8	4.1	3.6	5.1	64
Dataset 3	LOAM	354.8	123.8	80.6	154.8	**3.2**	-	-	-	-	-
	LOAM + RO	**88.4**	**13.2**	**6.8**	**21.8**	2.9	28.1	4.6	3.6	6.5	54
	R-LOAM	69.7	16.9	15.3	19.4	**3.0**	-	-	-	-	-
	R-LOAM + RO	**54.8**	**9.3**	**6.7**	**12.2**	**3.0**	23.9	5.3	3.9	7.2	58

Table 5.7 by means of APE. Two types of APE are presented: *APE mapping* is computed using the map-optimized poses of the LOAM algorithm. The results of *APE mapping* are *not* from the PG-optimized poses. It is important to note that the improvement when enabling the extension solely comes from the improved map quality and drift correction after successful TMOs. The map and drift are corrected by the map correction module (see Fig. 5.1). The reason to show the APE of map-optimized poses is simply that these poses are available online and can be used for real-time robot control. The PG-optimized poses are only used to correct drift in the map but have no immediate advantage for real-time robot control. *APE TMO* shows the average error of the poses used for TMO, i.e., the confirmed candidate poses in the sequences, which are added to the pose graph with a high-confidence constraint. Hence, these poses survived the candidate evaluation step and should have a low error.

In each of the three datasets, over 50 TMOs were performed. The median and mean APE TMO amounts to $< 6\,\mathrm{cm}$, which is desirable as constraints for PGO. However, when looking at the max. APE TMO, one notices that the error is much higher than the median or mean APE TMO. Nevertheless, it barely influences the performance. If the number of successful TMOs is high and regular, the drift introduced by a (slightly) wrong TMO can be corrected with the next one. However, in this short sequence between the TMOs, the map quality may suffer. This is due to the subgraph approach, where only the trajectory since the previous TMO is corrected during PGO. The table also shows that R-LOAM consistently performs better than LOAM. The latter suffers from early wrong scan matches, resulting in heavy drift. The proposed extension improves LOAM and R-LOAM performance for all datasets. The median APE of LOAM + RO amounts to less than 10% compared to state-of-the-art LOAM.



(a) R-LOAM (b) R-LOAM + RO

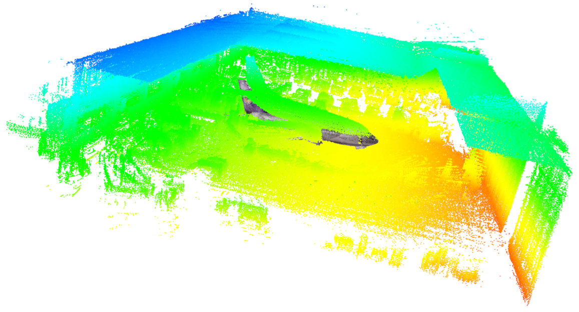
Figure 5.11: APE mapped onto the trajectory of Dataset 1 using R-LOAM (a) and R-LOAM + RO (b). The green circles in (b) mark the poses of the TMOs. The left red circles highlight a reduction in APE for R-LOAM + RO due to the TMOs. The zoomed-in red circle highlights a small part where R-LOAM performs better.

Despite the significantly lower APE of R-LOAM compared to LOAM, activating the extension for R-LOAM even further reduces the APE. Generally, it can be seen that the average mapping frequency is 3 Hz. This means that 3 out of 10 LiDAR scans are map-optimized and inserted into the map. Activating the extension slightly reduces the mapping frequency. This is due to the higher CPU load since the LOAM algorithm was executed on the same server as the extension. In a remote SLAM setup, the LOAM algorithm can run on the robotic platform and the proposed reference object-based TMO extension can run on a high-performance Edge Cloud server.

To summarize the findings, activating the reference object-based TMO extension can significantly increase localization and mapping accuracy. The results on three datasets in a visual airplane inspection scenario showed an improvement for LOAM and R-LOAM. The proposed EKF-based motion prior filtering step successfully filtered out unsuitable scan-to-model aligned poses for TMO.

Figure 5.11 visualizes the APE mapped onto the trajectory of R-LOAM (a) and R-LOAM + RO (b). It can be seen that especially in the rear of the airplane R-LOAM has an increased error, as marked by the red circle. R-LOAM + RO is able to perform TMOs in that region (green circles) and corrects accumulated drift.

However, the zoomed-in version in the right red circle of R-LOAM + RO shows an increased error compared to R-LOAM. Let's first take as an example the conventional LOAM algorithm, which builds a fully tilted map of a building, e.g., a vertical instead of a horizontal map due to early wrong scan matches. Now assume that all following scans of the dataset perfectly fit into this tilted map. This will create a nearly perfect, but tilted 3D representation. However, the APE will be very high, since an actual forward trajectory would be estimated as upward or downward. Now consider a trajectory, which is continuously improved by reference object-based TMOs. Due to the knowledge about the absolute pose of the reference object, the PG-optimized trajectory will have a low APE. However, correcting


(a) LOAM

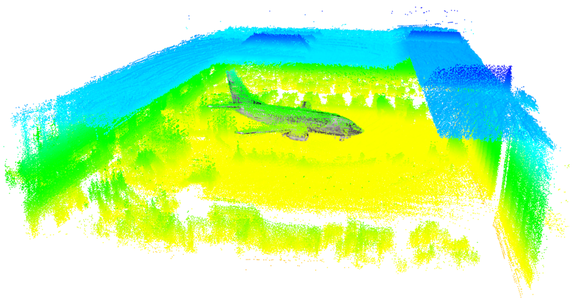

(b) LOAM + RO

Figure 5.12: Final feature maps of Dataset 1 using LOAM (a) and LOAM + RO (b). The feature maps are color-coded according to the Z-value. The airplane as a reference object is shown at the same initially estimated position in both figures. A wall and parts of the roof are removed for illustrative purposes.

a map with PG-optimized poses is not trivial. The PGO optimizes the pose graph according to the constraints. In the proposed approach, all graph nodes were added with edges of equal confidence in the information matrix. While this generally may decrease the APE, re-inserting or correcting the scans into the feature maps with these PG-optimized poses may lead to a lower cohesion. The cohesion of a point cloud can be understood as a measure of its compactness or integrity. By the map optimization process with a nonlinear least-squares method, the scan is fit into the map in the best possible way. This essentially also maximizes the cohesion of the feature maps. The PGO, however, aims at minimizing the cost as defined by the graph constraints. This mainly focuses on minimizing the APE but may neglect how well the scans actually fit into the map. Exactly this is the reason why the cohesion of the feature maps may decrease when correcting the scans with the PG-optimized poses. The map correction module as described in Section 5.2.5 implements a naïve approach. All scans are re-inserted into the map with the PG-optimized poses. Future work may improve map cohesion after TMOs, e.g., by scan rejection methods or additional scan-to-map optimization.

In fact, Razlaw et al. [69] experienced the inverse of the phenomenon. They found that by further refining poses from visual odometry with scan registration methods, i.e., ICP, the pose error is not further reduced. Instead, the map integrity is increased and the Mean Map Entropy (MME) is reduced.

The final feature maps of the conventional LOAM and LOAM + RO algorithms using Dataset 1 can be seen in Figure 5.12. The 3D model of the airplane is shown at the exact same position in both figures. In (b) it can be seen, that the final feature maps overlay very well with the 3D model. This would not be the case without the proper relative pose estimation between the UAV and the airplane before the exploration as explained in Section 5.3.4. In comparison, the map of LOAM in (a) appears tilted. Also, the mapped airplane does not overlay well with the actual position of the 3D model. LOAM + RO takes the reference model into account and performs regular TMOs whenever possible.

5.5 Chapter summary

This chapter proposed a novel extension to the LOAM algorithm leveraging an a priori known reference object to perform trajectory and map optimization (TMO) for 3D LiDAR-SLAM. For this, the geometry and pose of the reference object need to be known. The latter can be estimated from the starting position of the robot. The proposed approach was termed RO-LOAM.

The basic idea of the proposed extension is to trigger regular scan-to-model alignments for trajectory and map optimization. For this, isolated scans are used, containing mainly points of the reference object. The corresponding map-optimized poses of the LOAM algorithm are used as initial guesses. An iterative ICP-based method refines these poses with scan-to-model alignments. Due to alignment ambiguities, the MSE metric can not be used to verify if the alignments resulted in a low pose error. Hence, an EKF-based motion prior filtering step was proposed. A sequence of model-aligned poses is used as input to the EKF. If the last model-aligned pose (= candidate for TMO) is close to the motion prior of the EKF, it follows the motion model of the previous model-aligned poses. Therefore, it is considered a successful model convergence with low APE. The candidate pose is then used to perform Pose Graph Optimization (PGO) of the latest subgraph, which essentially corrects the trajectory since the last TMO. A map correction module then re-inserts the scan features with the corrected PG-optimized poses to correct drift and improve map quality. While all other modules of the extension are independent of the SLAM algorithm, the map correction module is highly dependent on the map structure and needs to be adapted accordingly.

The core advantage in comparison to R-LOAM is that the proposed extension can run in a remote SLAM setup, e.g., on an Edge Cloud. This allows for an increased number of scan-to-model alignment iterations. In contrast, R-LOAM tightly couples the 3D model into the map optimization process, which limits the number of iterations to a minimum for online performance. Most of the modules of RO-LOAM are decoupled from the LOAM algorithm, which makes it suitable to attach to other LiDAR-SLAM algorithms.

For the experiments, a visual airplane inspection scenario inside a hangar was selected. An octocopter UAV was equipped with an actuated VLP-16 LiDAR sensor, which was continuously rotated back and forth around the roll axis. LiDAR data was stored on an Intel NUC. Three datasets were recorded while the UAV was manually controlled by an operator. A Leica Total Station was employed to generate a highly detailed 3D model of the airplane as a reference object. The same system was used to track a prism on the UAV for GT comparison. By relatively estimating the pose between the start position of the UAV and the airplane, the SLAM and GT trajectories could be aligned. The same relative pose was used as the a priori position of the reference object for the proposed extension. A detailed description of the timestamp synchronization between the SLAM and GT system using an MSE-based sliding window approach was given.

The first experiments dealt with the calibration of the parameters, such as scan-to-model sequence length M, the buffer size L, #points used for the alignment, and #iter defining the number of model alignment iterations. The proposed extension has shown improvements for LOAM and R-LOAM on all three datasets by means of Absolute Pose Error (APE). In the experiments, the EKF-based filtering step ensured that only poses of well-converged scans were used for TMOs. The median APE of all TMOs amounted to $< 5\,\mathrm{cm}$.

To summarize, the proposed reference object-based trajectory and map optimization extension can improve 3D LiDAR-SLAM performance and even further reduce the APE when combined with R-LOAM. Future work can even further robustify the motion prior filtering step and map correction process.

Chapter 6

Conclusion

This chapter concludes this thesis by first summarizing the proposed methods. Finally, possibilities are discussed to achieve further improvements in future work.

6.1 Summary

Robotic platforms are increasingly used for search and rescue, disaster recovery, or surveillance operations, which might be dangerous for human beings. But they are also used to speed up or simplify tasks, such as manipulation or maintenance operations. Especially for visual inspection services, air-born robotic platforms have received increasing attention. Inspections of bridges, buildings, and also airplanes require accurate planning to access and inspect even hard-to-reach areas. Often, scaffolding is needed. In recent years, UAVs have become increasingly popular for inspection services. Improvements in battery capacity, weight reduction, and more efficient rotors have contributed to their rising success. However, GPS-based localization close to buildings may be inaccurate or, inside a building, no GPS may be available. Hence, UAVs are typically equipped with LiDAR sensors to detect obstacles, avoid collision and at the same time localize themselves with SLAM algorithms. To take high-resolution images of potential damages, UAVs have to get as close as possible to the object. For autonomous UAVs, this requires highly accurate localization and navigation. A crash into the inspection object may cause severe costs, especially in the case of airplane inspection.

To this end, three approaches have been proposed to improve localization accuracy and mapping quality for 3D LiDAR-SLAM. To achieve this, the well-known LOAM framework [12] was modified and extended. Prior knowledge has been leveraged in related work in the form of a priori known maps, 2D architectural floor plans or emergency maps, 3D CAD models of a building and workpiece, or also aerial images. However, previous work either uses prior knowledge only for 2D LiDAR-SLAM or converts 3D CAD models to a point-sampled initial map for LiDAR-SLAM. The latter is equivalent to having a high-accuracy map available before the exploration. In practice, this can also be achieved with a separate exploration run, mapping the environment with high accuracy sensors.

Chapter 3 introduced a modification of the LOAM framework to enable immutable initial map creation, termed Init-LOAM. The core idea is to create an initial map at the starting position of the robot with an actuated 3D LiDAR sensor. This initial map remains unmodified during the exploration. The motivation is that an initial map can be created with much higher accuracy at the starting position of the robot and that a modification of the initial map during the exploration increases localization error. The method assumes a static environment. To create the initial map, an initialization phase is added to the LOAM pipeline. In this phase, captured LiDAR scans are transformed to the map frame with the actuator readings. A method was proposed to keep the voxelized features of the initial map separate from the features of the dynamic map. The latter includes the scan features acquired during exploration. Detailed algorithm descriptions were given to implement Init-LOAM as a modification of the A-LOAM open-source framework. In a simulated environment, three datasets of one indoor and two outdoor scenarios were generated. A virtual 3D LiDAR sensor was mounted on a quadcopter UAV and continuously rotated back and forth with an actuator during the initialization and exploration phases. The first experimental results investigated the effect of voxelization and measurement imprecision on surface representations. As measurement imprecision, ranging errors and actuator reading errors were considered. Results were presented for a planar and curved surface. It has been shown that with realistic errors, a higher number of sampled points on a planar surface lets the voxel centroid of the distorted points converge towards the real centroid. However, for a curved surface not even the real centroid lies on the surface itself. Using a triangular mesh, a higher number of LiDAR sweeps showed an increased RMSE. However, no significant influence on the APE using a dataset could be found. The proposed method has proven robust against actuator imprecision up to $\sigma \leq 3.0$ degrees. Using all three datasets, Init-LOAM with the immutable initial map showed superior performance over Init-LOAM with a variable initial map, and over conventional LOAM. Hence, it can be concluded that in a static environment, leaving the initial map unmodified during the exploration can improve localization and mapping performance.

The other proposed methods do not require a separate exploration and seamlessly integrate prior knowledge. In contrast to related work, only a reference object in the environment is leveraged, which does not need to be sufficient for a localization-only approach. No assumptions about the environment are made. Also, related work directly integrates point-sampled CAD models into the map for localization. Many SLAM algorithms use down-sampled or voxelized map representations to overcome growing map sizes and to maintain real-time performance. The proposed methods in this thesis aim at using highly dense point clouds or directly integrating a CAD triangular mesh of the reference object into the LOAM pipeline, which is operating internally with a voxelized map.

Chapter 4 proposed R-LOAM, a modification of the conventional LOAM algorithm to tightly couple a 3D triangular mesh into the map optimization process. For this, point-to-mesh correspondences are used in addition to conventional point-to-point correspondences. The method assumes the geometry and 6-DoF pose of the reference object in a global coordinate frame to be known. To form point-to-mesh correspondences, virtual points on the reference object's surface are computed. For this, the use of an AABB tree was proposed to accelerate closest-point computation. A map optimization formulation was presented, which jointly optimizes conventional point-to-point and point-to-mesh correspondences. Two indoor and one outdoor visual inspection scenarios were simulated, using an airplane, a small van, and the Eiffel Tower as reference objects. A virtual VLP-16 and OS1-128 were used to generate LiDAR data. Results have shown that a logarithmic weight increase for mesh features with the number of map optimization iterations shows the best performance. For all three datasets, R-LOAM resulted in a reduced APE and RE. Using the OS1-128 was essential to detect sufficient points on the surface of the van as a reference object. Due to the higher number of extracted features, the OS1-128 resulted in consistently reduced APE compared to the VLP-16. R-LOAM was able to achieve a subcentimeter median APE at the Eiffel Tower with the VLP-16 at 35 iterations, whereas conventional LOAM failed. Generally, a higher number of map optimization iterations has proven beneficial for R-LOAM. The map and dense 3D reconstruction quality are tightly coupled to the localization error. Illustrations have demonstrated the improved reconstruction accuracy of the proposed R-LOAM algorithm. To summarize, taking the reference object into account can significantly reduce the localization error and at the same time improve the map and reconstruction quality.

Chapter 5 proposed an extension to LiDAR-SLAM algorithms termed RO-LOAM. It leverages a 3D reference object for trajectory and map optimization (TMO). Instead of tightly coupling the reference object into the LOAM pipeline as in R-LOAM, a fully parallelized extension was proposed with the possibility to Edge Cloud processing. The core idea is to register a sequence of scans to the 3D model with a high number of iterations, refining the map-optimized poses, which are used as initial guesses. However, the MSE metric is not suitable to determine if the alignment is successful. Hence, an EKF-based motion prior filtering step is employed to determine if the last pose in the sequence is suitable for TMO. If successful, the pose is added to a pose graph with a high-confidence constraint, followed by a Pose Graph Optimization (PGO). The corrected poses are then used to rebuild the map, effectively reducing drift and improving map quality. Only the map correction module is tightly coupled to the LOAM algorithm, which makes the rest of the proposed extension suitable to attach to other LiDAR-SLAM algorithms. For the experimental setup, a Leica MultiStation was used to generate a highly accurate 3D model of a B737 airplane as a reference object. An octocopter UAV carried an actuated VLP-16 LiDAR. A Leica prism was mounted on a rod to be tracked by the Leica system for ground-truth measurements. The UAV was manually controlled by an operator inside a hangar, following three trajectories for potential visual inspection along one side of the airplane. A scan-to-model alignment approach for the relative pose estimation between the starting position of the UAV and the airplane as a reference object was explained. Also, a detailed description of the timestamp synchronization between the SLAM and the ground-truth system was given. The results have shown an improvement when enabling the proposed extension for LOAM and R-LOAM on all three datasets. Even for R-LOAM, which uses the same 3D model, a further improvement could be seen with the extension. A major advantage of the proposed method is the possibility for offloaded processing to an Edge Cloud and the higher number of iterations compared to R-LOAM. Ultimately, LOAM + RO reduced the median APE to $< 10\%$ compared to LOAM. R-LOAM + RO has shown a reduction to $< 55\%$ compared to R-LOAM. To summarize, despite the conceptual similarity of R-LOAM and the proposed TMO extension, the parallelized offloaded processing allows for a more accurate scan-to-model convergence and a lower APE.

6.2 Outlook

The proposed methods in this thesis have shown that leveraging a known 3D reference object can indeed improve LOAM accuracy. However, certain assumptions were made, which could be addressed by future work:

The position of the reference object is known or can be estimated from the starting position. In the experimental setups presented in this thesis, the position of the reference object was exactly known in a global coordinate frame or could be estimated from the starting position of the robotic platform. In practice, the exact relative pose is never known, since the robot can be placed arbitrarily. Hence, global relocalization or registration methods are required to get a first coarse estimate of the relative pose. This can be treated as a "kidnapped robot" problem. After the coarse relocalization, fine registration, e.g., with the proposed scan-to-model alignments can be conducted. However, this still assumes that at least some parts of the reference object are visible from the starting position of the robot. Future work could further develop the proposed methods to not make the assumption of an initially known position of the reference object. For example, the knowledge that in the next hangar a known airplane is placed could be already leveraged to improve LOAM accuracy. The robot starts in another hangar with conventional LOAM. Once the known airplane is detected, the position in the map is determined and the reference object is integrated into the localization and mapping process. This may not improve the APE, since drift may have already occurred until the robot has entered the next hangar and started leveraging the reference object. Nevertheless, additional drift can be corrected for as long as the robot is in the vicinity of the reference object.

The environment is static. The environments used in this thesis were mostly static. Inside a hangar, usually, multiple airplanes are maintained at the same time. Also, maintenance engineers may drive vehicles or ground staff walks between the airplanes. Airplanes may be moved out of or into the hangar. All these situations introduce dynamics into the inspection scenario. Especially large changes in the scenery, i.e., opening large hangar doors may affect LOAM accuracy during an airplane inspection with a UAV. However, the proposed R-LOAM and RO-LOAM algorithms only assume an isolated reference object. Big changes in the environment far from the reference object only affect the conventional LOAM pipeline. However, dynamic objects close to the reference object may indeed influence the proposed approaches. Specifically, the scan isolation method employed a naïve bounding box cropping method. More advanced isolated methods, e.g., convex hull cropping could be applied instead. Alternatively, future work could investigate correspondence rejection methods to identify wrong point-to-mesh correspondences.

There is only low drift to be compensated by R-LOAM and RO-LOAM. One certain limitation of the proposed approaches is the capability of correcting low drift only. This is due to the nature of the point-to-model correspondences using the map-optimized transform as an initial guess. If a larger drift exists, wrong correspondences may let the optimization diverge from the correct solution. Also, with a large drift, the scan isolation module would remove many points actually belonging to the reference model. The proposed methods assume a significant Intersection over Union (IoU) between the estimated and actual bounding boxes of the reference object. Future work could investigate if the scan isolation module can be replaced with smarter maximum correspondence thresholds or outlier rejection methods.

Before the estimation of correspondences, global relocalization methods could be employed to find a good initial estimate for the ICP-based scan-to-model alignment method. Since for global registration/relocalization one can not make any assumption about the initial guess, the current scan would need to be matched against the map. Global relocalization with the use of the reference object is challenging because it is fully unknown which scan points belong to the reference object and which belong to the surrounding environment.

The 3D geometry of the reference object is known. The proposed methods either assume an exact 3D CAD mesh (R-LOAM), or a highly accurate 3D point cloud (RO-LOAM) of the reference object. In airplane inspections, scenarios may exist where the reference object temporarily changes its appearance. For example, maintenance engineers may have set up scaffolding partially around the airplane, the flaps are lowered or cables/pipes are connected. These close-by objects or changes in appearance may have a significant effect on the proposed methods.

Let's assume a visual inspection scenario of a building, which suffered damage on parts of the outer structure. With an accurate 3D model, the proposed methods can be employed until the damages on the facade appear in the scans. Future works could investigate how to correctly identify the deviations, e.g., in the correspondences, or even to update the 3D model. However, the latter may lead to inaccurate 3D models, which even result in a decreased performance compared to conventional 3D LiDAR-SLAM algorithms. Here, future work could investigate, if already proposed methods for map maintenance in long-term localization and mapping can be applied.

Bibliography

Publications by the author

Journal publications

[1] M. Oelsch, M. Karimi, and E. Steinbach, “R-loam: Improving lidar odometry and mapping with point-to-mesh features of a known 3d reference object,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2068–2075, 2021. DOI: 10.1109/LRA.2021.3060413.

[2] ——, “Ro-loam: 3d reference object-based trajectory and map optimization in lidar odometry and mapping,” *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 6806–6813, 2022. DOI: 10.1109/LRA.2022.3177846.

[3] M. Karimi, M. Oelsch, O. Stengel, E. Babaians, and E. Steinbach, “Lola-slam: Low-latency lidar slam using continuous scan slicing,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2248–2255, 2021. DOI: 10.1109/LRA.2021.3060721.

[4] M. Karimi, E. Babaians, M. Oelsch, and E. Steinbach, “Deep fusion of a skewed redundant magnetic and inertial sensor for heading state estimation in a saturated indoor environment,” *International Journal of Semantic Computing*, vol. 15, no. 03, pp. 313–335, 2021. DOI: 10.1142/S1793351X21400079. eprint: https://doi.org/10.1142/S1793351X21400079. [Online]. Available: https://doi.org/10.1142/S1793351X21400079.

Conference publications

[5] M. Oelsch, M. Karimi, and E. Steinbach, “Init-loam: Lidar-based localization and mapping with a static self-generated initial map,” in *2021 20th International Conference on Advanced Robotics (ICAR)*, 2021, pp. 865–872. DOI: 10.1109/ICAR53236.2021.9659358.

[6] M. Oelsch, D. Van Opdenbosch, and E. Steinbach, “Survey of visual feature extraction algorithms in a mars-like environment,” in *2017 IEEE International Symposium on Multimedia (ISM)*, 2017, pp. 322–325. DOI: 10.1109/ISM.2017.58.

[7] M. Oelsch, B. Gulecyuz, and E. Steinbach, “Mid: A novel contrast metric for the mser detector,” in *2018 IEEE International Symposium on Multimedia (ISM)*, 2018, pp. 29–35. DOI: 10.1109/ISM.2018.00014.

[8] M. Karimi, E. Babaians, M. Oelsch, T. Aykut, and E. Steinbach, "Skewed-redundant hall-effect magnetic sensor fusion for perturbation-free indoor heading estimation," in *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*, 2020, pp. 367–374. DOI: 10.1109/IRC.2020.00064.

[9] D. Van Opdenbosch, M. Oelsch, A. Garcea, T. Aykut, and E. Steinbach, "Selection and compression of local binary features for remote visual slam," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 7270–7277. DOI: 10.1109/ICRA.2018.8463202.

[10] D. Van Opdenbosch, M. Oelsch, A. Garcea, and E. Steinbach, "A joint compression scheme for local binary feature descriptors and their corresponding bag-of-words representation," in *2017 IEEE Visual Communications and Image Processing (VCIP)*, 2017, pp. 1–4. DOI: 10.1109/VCIP.2017.8305155.

[11] M. Tappe, D. Dose, M. Oelsch, M. Karimi, L. Hösch, L. Heller, J. Knufinke, M. Alpen, J. Horn, and C. Bachmeir, "UAS based autonomous visual inspection of airplane surface defects," in *NDE 4.0, Predictive Maintenance, and Communication and Energy Systems in a Globally Networked World*, N. G. Meyendorf, S. Farhangdoust, and C. Niezrecki, Eds., International Society for Optics and Photonics, vol. 12049, SPIE, 2022, pp. 8 –21. DOI: 10.1117/12.2612579. [Online]. Available: https://doi.org/10.1117/12.2612579.

General publications

[12] J. Zhang and S. Singh, "LOAM: Lidar Odometry and Mapping in Real-time," *Autonomous Robots*, vol. 41, no. 2, pp. 401–416, 2017, ISSN: 15737527. DOI: 10.1007/s10514-016-9548-2.

[13] F. Lourenço and H. Araujo, "Intel realsense sr305, d415 and l515: Experimental evaluation and comparison of depth estimation," in *Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 4: VISAPP,*, INSTICC, SciTePress, 2021, pp. 362–369, ISBN: 978-989-758-488-6. DOI: 10.5220/0010254203620369.

[14] Velodyne LiDAR Inc., "Velodyne VLP-16 datasheet," Tech. Rep., 2017, p. 2.

[15] Ouster Inc., "Ouster OS1 datasheet," Tech. Rep., 2021, p. 5.

[16] R. B. Rusu, "Semantic 3d object maps for everyday manipulation in human living environments," PhD thesis, Computer Science department, Technische Universitaet Muenchen, Germany, 2009.

[17] M. Kazhdan, M. Chuang, S. Rusinkiewicz, and H. Hoppe, "Poisson Surface Reconstruction with Envelope Constraints," *Computer Graphics Forum*, vol. 39, no. 5, pp. 173–182, 2020, ISSN: 14678659. DOI: 10.1111/cgf.14077.

[18] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz, "Aligning point cloud views using persistent feature histograms," *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pp. 3384–3391, 2008. DOI: 10.1109/IROS.2008.4650967.

[19] R. B. Rusu and S. Cousins, “3D is here: Point Cloud Library (PCL),” in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, 2011.

[20] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, “MeshLab: an Open-Source Mesh Processing Tool,” in *Eurographics Italian Chapter Conference*, V. Scarano, R. D. Chiara, and U. Erra, Eds., The Eurographics Association, 2008, ISBN: 978-3-905673-68-5. DOI: 10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136.

[21] D. Allegra, G. Gallo, L. Inzerillo, M. Lombardo, F. L. M. Milotta, C. Santagati, and F. Stanco, “Hand held 3D scanning for cultural heritage,” in *Advances in Religious and Cultural Studies*, IGI Global, 2017, pp. 475–499.

[22] A. Jacobson, D. Panozzo, *et al.*, *libigl: A simple C++ geometry processing library*, 2018. [Online]. Available: https://libigl.github.io/.

[23] The CGAL Project, *CGAL User and Reference Manual*, 5.4. CGAL Editorial Board, 2022. [Online]. Available: https://doc.cgal.org/5.4/Manual/packages.html.

[24] Q.-Y. Zhou, J. Park, and V. Koltun, “Open3D: A modern library for 3D data processing,” *arXiv:1801.09847*, 2018.

[25] R. B. Rusu, N. Blodow, and M. Beetz, “Fast Point Feature Histograms (FPFH) for 3D Registration,” *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3212–3217, 2009, ISSN: 10504729. DOI: 10.1109/ROBOT.2009.5152473.

[26] C. Choy, J. Park, and V. Koltun, “Fully convolutional geometric features,” *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2019-October, pp. 8957–8965, 2019, ISSN: 15505499. DOI: 10.1109/ICCV.2019.00905.

[27] D. Droeschel and S. Behnke, “Omnidirectional Perception for Lightweight MAVs using a Continuously Rotating 3D Laser,” *Photogrammetrie - Fernerkundung - Geoinformation*, vol. 2014, no. 5, pp. 451–464, 2014, ISSN: 14328364. [Online]. Available: http://www.ais.uni-bonn.de/papers/PFG_2014_Droeschel.pdf%5Cnhttp://openurl.ingenta.com/content/xref?genre=article&issn=1432-8364&volume=2014&issue=5&spage=451.

[28] D. Chetverikov, D. Svirko, D. Stepanov, and P. Krsek, “The trimmed iterative closest point algorithm,” *Proceedings - International Conference on Pattern Recognition*, vol. 16, no. 3, pp. 545–548, 2002, ISSN: 10514651. DOI: 10.1109/icpr.2002.1047997.

[29] G. C. Sharp, S. W. Lee, and D. K. Wehe, “ICP registration using invariant features,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 1, pp. 90–102, 2002, ISSN: 01628828. DOI: 10.1109/34.982886.

[30] Y. Chen and G. Medioni, “Object modeling by registration of multiple range images,” in *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, 1991, 2724–2729 vol.3. DOI: 10.1109/ROBOT.1991.132043.

[31] A. Segal, D. Haehnel, and S. Thrun, “Generalized-icp.,” in *Robotics: science and systems*, Seattle, WA, vol. 2, 2009, p. 435.

[32] S. Rusinkiewicz, "A symmetric objective function for icp," *ACM Trans. Graph.*, vol. 38, no. 4, 2019, ISSN: 0730-0301. DOI: 10.1145/3306346.3323037. [Online]. Available: https://doi.org/10.1145/3306346.3323037.

[33] M. Lamine Tazir, T. Gokhool, P. Checchin, L. Malaterre, and L. Trassoudaine, "CICP: Cluster Iterative Closest Point for sparse–dense point cloud registration," *Robotics and Autonomous Systems*, vol. 108, pp. 66–86, 2018, ISSN: 09218890. DOI: 10.1016/j.robot.2018.07.003.

[34] J. Yang, H. Li, D. Campbell, and Y. Jia, "Go-ICP: A Globally Optimal Solution to 3D ICP Point-Set Registration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 11, pp. 2241–2254, 2016, ISSN: 01628828. DOI: 10.1109/TPAMI.2015.2513405. arXiv: 1605.03344.

[35] Y. Wang and J. Solomon, "Deep closest point: Learning representations for point cloud registration," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2019-October, pp. 3522–3531, 2019, ISSN: 15505499. DOI: 10.1109/ICCV.2019.00362. arXiv: 1905.03304.

[36] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph Cnn for learning on point clouds," *ACM Transactions on Graphics*, vol. 38, no. 5, Article 146, 2019, ISSN: 15577368. DOI: 10.1145/3326362. arXiv: 1801.07829.

[37] Q. Y. Zhou, J. Park, and V. Koltun, "Fast global registration," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9906 LNCS, pp. 766–782, 2016, ISSN: 16113349. DOI: 10.1007/978-3-319-46475-6_47.

[38] Y. Aoki, H. Goforth, R. A. Srivatsan, and S. Lucey, "Pointnetlk: Robust & efficient point cloud registration using pointnet," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7163–7172.

[39] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-January, pp. 77–85, 2017. DOI: 10.1109/CVPR.2017.16. arXiv: 1612.00593.

[40] H. Yang, J. Shi, and L. Carlone, "Teaser: Fast and certifiable point cloud registration," *IEEE Transactions on Robotics*, vol. 37, no. 2, pp. 314–333, 2021, ISSN: 19410468. DOI: 10.1109/TRO.2020.3033695. arXiv: 2001.07715.

[41] H. Yin, L. Tang, X. DIng, Y. Wang, and R. Xiong, "LocNet: Global Localization in 3D Point Clouds for Mobile Vehicles," *IEEE Intelligent Vehicles Symposium, Proceedings*, vol. 2018-June, no. Iv, pp. 728–733, 2018. DOI: 10.1109/IVS.2018.8500682. arXiv: 1712.02165.

[42] P. J. Schneider and D. H. Eberly, "Chapter 11 - intersection in 3d," in *Geometric Tools for Computer Graphics*, ser. The Morgan Kaufmann Series in Computer Graphics, P. J. Schneider and D. H. Eberly, Eds., San Francisco: Morgan Kaufmann, 2003, pp. 481–662, ISBN: 978-1-55860-594-7. DOI: https://doi.org/10.1016/B978-155860594-7/50014-X. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B978155860594750014X.

[43] C. Ericson, "Chapter 3 - a math and geometry primer," in *Real-Time Collision Detection*, ser. The Morgan Kaufmann Series in Interactive 3D Technology, C. Ericson, Ed., San Francisco: Morgan Kaufmann, 2005, pp. 23–73, ISBN: 978-1-55860-732-3. DOI: https://doi.org/10.1016/B978-1-55860-732-3.50008-5. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9781558607323500085.

[44] ——, "Chapter 6 - bounding volume hierarchies," in *Real-Time Collision Detection*, ser. The Morgan Kaufmann Series in Interactive 3D Technology, C. Ericson, Ed., San Francisco: Morgan Kaufmann, 2005, pp. 235–284, ISBN: 978-1-55860-732-3. DOI: https://doi.org/10.1016/B978-1-55860-732-3.50011-5. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9781558607323500115.

[45] G. van den Bergen, "Efficient Collision Detection of Complex Deformable Models using AABB Trees," *Graphics Tools—The jgt Editors' Choice*, pp. 131–144, 2005. DOI: 10.1201/b10628-18.

[46] "The morgan kaufmann series in interactive 3d technology," in *Real-Time Collision Detection*, ser. The Morgan Kaufmann Series in Interactive 3D Technology, C. Ericson, Ed., San Francisco: Morgan Kaufmann, 2005, p. iv, ISBN: 978-1-55860-732-3. DOI: https://doi.org/10.1016/B978-1-55860-732-3.50021-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9781558607323500218.

[47] C. Ericson, "Chapter 5 - basic primitive tests," in *Real-Time Collision Detection*, ser. The Morgan Kaufmann Series in Interactive 3D Technology, C. Ericson, Ed., San Francisco: Morgan Kaufmann, 2005, pp. 125–233, ISBN: 978-1-55860-732-3. DOI: https://doi.org/10.1016/B978-1-55860-732-3.50010-3. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9781558607323500103.

[48] F. Bosché, "Plane-based registration of construction laser scans with 3D/4D building models," *Advanced Engineering Informatics*, vol. 26, no. 1, pp. 90–102, 2012, ISSN: 14740346. DOI: 10.1016/j.aei.2011.08.009.

[49] S. Umeyama, "Least-Squares Estimation of Transformation Parameters Between Two Point Patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 4, pp. 376–380, 1991, ISSN: 01628828. DOI: 10.1109/34.88573.

[50] B. K. P. Horn, H. M. Hilden, and S. Negahdaripour, "Closed-form solution of absolute orientation using orthonormal matrices," *Journal of the Optical Society of America A*, vol. 5, no. 7, p. 1127, 1988, ISSN: 1084-7529. DOI: 10.1364/josaa.5.001127.

[51] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner, “On measuring the accuracy of slam algorithms,” *Autonomous Robots*, vol. 27, pp. 387–407, 2009.

[52] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, “A benchmark for the evaluation of RGB-D SLAM systems,” *IEEE International Conference on Intelligent Robots and Systems*, pp. 573–580, 2012, ISSN: 21530858. DOI: 10.1109/IROS.2012.6385773.

[53] F. Henrik, “Fusion of imu and monocular-slam in a loosely coupled ekf,” Master’s thesis, Linköping University, Automatic Control, 2017, p. 61.

[54] B. Das, G. Dobie, and S. G. Pierce, “AS-EKF: A Delay Aware State Estimation Technique for Telepresence Robot Navigation,” *Proceedings - 3rd IEEE International Conference on Robotic Computing, IRC 2019*, pp. 624–629, 2019. DOI: 10.1109/IRC.2019.00126.

[55] R. Van Der Merwe, E. Wan, and S. Julier, “Sigma-point kalman filters for nonlinear estimation and sensor-fusion: Applications to integrated navigation,” in *Aiaa guidance, navigation, and control conference and exhibit*, 2004, p. 5120.

[56] F. Lu and E. Milios, “Globally consistent range scan alignment for environment mapping,” *Autonomous robots*, vol. 4, no. 4, pp. 333–349, 1997.

[57] L. Pfotzer, J. Oberlaender, A. Roennau, and R. Dillmann, “Development and calibration of karola, a compact, high-resolution 3d laser scanner,” in *2014 IEEE International Symposium on Safety, Security, and Rescue Robotics (2014)*, IEEE, 2014, pp. 1–6.

[58] M. Bosse and R. Zlot, “Continuous 3D scan-matching with a spinning 2D laser,” pp. 4312–4319, 2009. DOI: 10.1109/robot.2009.5152851.

[59] M. Bosse, R. Zlot, and P. Flick, “Zebedee: Design of a spring-mounted 3-D range sensor with application to mobile mapping,” *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1104–1119, 2012, ISSN: 15523098. DOI: 10.1109/TRO.2012.2200990.

[60] M. Schadler, J. Stuckler, and S. Behnke, “Multi-resolution surfel mapping and real-time pose tracking using a continuously rotating 2D laser scanner,” *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR 2013*, pp. 5–10, 2013. DOI: 10.1109/SSRR.2013.6719373.

[61] J. Stückler and S. Behnke, “Multi-resolution surfel maps for efficient dense 3D modeling and tracking,” *Journal of Visual Communication and Image Representation*, vol. 25, no. 1, pp. 137–147, 2014, ISSN: 10473203. DOI: 10.1016/j.jvcir.2013.02.008.

[62] D. Droeschel, M. Nieuwenhuisen, M. Beul, D. Holz, J. Stückler, and S. Behnke, “Multilayered Mapping and Navigation for Autonomous Micro Aerial Vehicles,” *Journal of Field Robotics*, vol. 33, no. 4, pp. 451–475, 2016, ISSN: 15564967. DOI: 10.1002/rob.21603.

[63] D. Droeschel, M. Schwarz, and S. Behnke, “Continuous mapping and localization for autonomous navigation in rough terrain using a 3d laser scanner,” *Robotics and Autonomous Systems*, vol. 88, pp. 104–115, 2017.

[64] S. Kohlbrecher, O. Von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation," *9th IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR 2011*, pp. 155–160, 2011. DOI: 10.1109/SSRR.2011.6106777.

[65] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: An open-source robot operating system," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, 2009.

[66] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2016, pp. 1271–1278.

[67] R. Opromolla, G. Fasano, G. Rufino, M. Grassi, and A. Savvaris, "LIDAR-inertial integration for UAV localization and mapping in complex environments," *2016 International Conference on Unmanned Aircraft Systems, ICUAS 2016*, pp. 649–656, 2016. DOI: 10.1109/ICUAS.2016.7502580.

[68] G. Ajay Kumar, A. K. Patil, R. Patil, S. S. Park, and Y. H. Chai, "A LiDAR and IMU integrated indoor navigation system for UAVs and its application in real-time pipeline classification," *Sensors (Switzerland)*, vol. 17, no. 6, 2017, ISSN: 14248220. DOI: 10.3390/s17061268.

[69] J. Razlaw, D. Droeschel, D. Holz, and S. Behnke, "Evaluation of registration methods for sparse 3d laser scans," in *2015 European Conference on Mobile Robots (ECMR)*, IEEE, 2015, pp. 1–7.

[70] R. Dubé, D. Dugas, E. Stumm, J. Nieto, R. Siegwart, and C. Cadena, "SegMatch: Segment based place recognition in 3D point clouds," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 5266–5272, 2017, ISSN: 10504729. DOI: 10.1109/ICRA.2017.7989618.

[71] R. Dubé, A. Cramariuc, D. Dugas, J. Nieto, R. Siegwart, and C. Cadena, "SegMap: 3d segment mapping using data-driven descriptors," in *Robotics: Science and Systems (RSS)*, 2018.

[72] D. Rozenberszki and A. L. Majdik, "LOL: Lidar-only Odometry and Localization in 3D point cloud maps," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2020, pp. 4379–4385, ISBN: 9781728173955. DOI: 10.1109/ICRA40945.2020.9197450. arXiv: 2007.01595.

[73] W. Zhen and S. Scherer, "A Unified 3D Mapping Framework Using a 3D or 2D LiDAR," *Springer Proceedings in Advanced Robotics*, vol. 11, pp. 702–711, 2020, ISSN: 25111264. DOI: 10.1007/978-3-030-33950-0_60. arXiv: 1810.12515.

[74] R. Dubé, A. Gawel, H. Sommer, J. Nieto, R. Siegwart, and C. Cadena, "An online multi-robot SLAM system for 3D LiDARs," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2017-Septe, pp. 1004–1011, 2017, ISSN: 21530866. DOI: 10.1109/IROS.2017.8202268.

[75] X. Ji, L. Zuo, C. Zhang, and Y. Liu, "LLOAM: LiDAR Odometry and Mapping with Loop-closure Detection Based Correction," *Proceedings of 2019 IEEE International Conference on Mechatronics and Automation, ICMA 2019*, pp. 2475–2480, 2019. DOI: 10.1109/ICMA.2019.8816388.

[76] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[77] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly of Applied Mathematics*, vol. 2, no. 2, pp. 164–168, 1944, ISSN: 0033569X, 15524485. [Online]. Available: http://www.jstor.org/stable/43633451.

[78] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963, ISSN: 03684245. [Online]. Available: http://www.jstor.org/stable/2098941.

[79] T. Shan and B. Englot, "LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain," *IEEE International Conference on Intelligent Robots and Systems*, pp. 4758–4765, 2018, ISSN: 21530866. DOI: 10.1109/IROS.2018.8594299.

[80] H. Wang, C. Wang, C.-L. Chen, and L. Xie, "F-loam : Fast lidar odometry and mapping," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 4390–4396. DOI: 10.1109/IROS51168.2021.9636655.

[81] D. Meyer-Delius, J. Hess, G. Grisetti, and W. Burgard, "Temporary maps for robust localization in semi-static environments," *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, pp. 5750–5755, 2010. DOI: 10.1109/IROS.2010.5648920.

[82] M. P. Parsley and S. J. Julier, "Exploiting prior information in GraphSLAM," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2638–2643, 2011, ISSN: 10504729. DOI: 10.1109/ICRA.2011.5979628.

[83] A. Walcott-Bryant, M. Kaess, H. Johannsson, and J. J. Leonard, "Dynamic pose graph SLAM: Long-term mapping in low dynamic environments," *IEEE International Conference on Intelligent Robots and Systems*, pp. 1871–1878, 2012, ISSN: 21530858. DOI: 10.1109/IROS.2012.6385561.

[84] G. D. Tipaldi, D. Meyer-Delius, and W. Burgard, "Lifelong localization in changing environments," *International Journal of Robotics Research*, vol. 32, no. 14, pp. 1662–1678, 2013, ISSN: 02783649. DOI: 10.1177/0278364913502830.

[85] J. Biswas and M. Veloso, "Episodic non-markov localization: Reasoning about short-term and long-term features," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 3969–3974. DOI: 10.1109/ICRA.2014.6907435.

[86] D. M. Rosen, J. Mason, and J. J. Leonard, "Towards lifelong feature-based mapping in semi-static environments," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2016-June, pp. 1063–1070, 2016, ISSN: 10504729. DOI: 10.1109/ICRA.2016.7487237.

[87] M. Fehr, F. Furrer, I. Dryanovski, J. Sturm, I. Gilitschenski, R. Siegwart, and C. Cadena, "TSDF-based change detection for consistent long-term dense reconstruction and dynamic object discovery," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 5237–5244, 2017, ISSN: 10504729. DOI: 10.1109/ICRA.2017.7989614.

[88] N. Shaik, T. Liebig, C. Kirsch, and H. Müller, "Dynamic map update of non-static facility logistics environment with a multi-robot system," in *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, Springer, 2017, pp. 249–261.

[89] R. Dubé, M. G. Gollub, H. Sommer, I. Gilitschenski, R. Siegwart, C. Cadena, and J. Nieto, "Incremental-segment-based localization in 3-d point clouds," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1832–1839, 2018, ISSN: 23773766. DOI: 10.1109/LRA.2018.2803213.

[90] P. Egger, P. V. Borges, G. Catt, A. Pfrunder, R. Siegwart, and R. Dubé, "PoseMap: Lifelong, Multi-Environment 3D LiDAR Localization," *IEEE International Conference on Intelligent Robots and Systems*, pp. 3430–3437, 2018, ISSN: 21530866. DOI: 10.1109/IROS.2018.8593854.

[91] A. Pfrunder, P. V. Borges, A. R. Romero, G. Catt, and A. Elfes, "Real-time autonomous ground vehicle navigation in heterogeneous environments using a 3D LiDAR," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2017-Septe, pp. 2601–2608, 2017, ISSN: 21530866. DOI: 10.1109/IROS.2017.8206083.

[92] X. Chen, T. Labe, L. Nardi, J. Behley, and C. Stachniss, "Learning an overlap-based observation model for 3D LiDAR localization," *IEEE International Conference on Intelligent Robots and Systems*, pp. 4602–4608, 2020, ISSN: 21530866. DOI: 10.1109/IROS45743.2020.9340769. arXiv: 2105.11717.

[93] W. Winterhalter, F. Fleckenstein, B. Steder, L. Spinello, and W. Burgard, "Accurate indoor localization for RGB-D smartphones and tablets given 2D floor plans," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2015-December, pp. 3138–3143, 2015, ISSN: 21530866. DOI: 10.1109/IROS.2015.7353811.

[94] F. Boniardi, T. Caselitz, R. Kümmerle, and W. Burgard, "Robust lidar-based localization in architectural floor plans," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 3318–3324.

[95] F. Boniardi, T. Caselitz, R. Kümmerle, and W. Burgard, "A pose graph-based localization system for long-term navigation in CAD floor plans," *Robotics and Autonomous Systems*, vol. 112, pp. 84–97, 2019, ISSN: 09218890. DOI: 10.1016/j.robot.2018.11.003.

[96] M. Mielle, M. Magnusson, and A. J. Lilienthal, "Ursim: Unique regions for sketch map interpretation and matching," *Robotics*, vol. 8, no. 2, 2019, ISSN: 2218-6581. DOI: 10.3390/robotics8020043. [Online]. Available: https://www.mdpi.com/2218-6581/8/2/43.

[97] M. Mielle, M. Magnusson, H. Andreasson, and A. J. Lilienthal, "SLAM auto-complete: Completing a robot map using an emergency map," *SSRR 2017 - 15th IEEE International Symposium on Safety, Security and Rescue Robotics, Conference*, pp. 35–40, 2017. DOI: 10.1109/SSRR.2017.8088137. arXiv: 1702.05087.

[98] M. Mielle, M. Magnusson, and A. J. Lilienthal, "The auto-complete graph: Merging and mutual correction of sensor and prior maps for slam," *Robotics*, vol. 8, no. 2, 2019, ISSN: 2218-6581. DOI: 10.3390/robotics8020040. [Online]. Available: https://www.mdpi.com/2218-6581/8/2/40.

[99] K. Müller, S. Bauer, J. Wasza, and J. Hornegger, "Automatic multi-modal tof/ct organ surface registration," in *Bildverarbeitung für die Medizin 2011: Algorithmen - Systeme - Anwendungen Proceedings des Workshops vom 20. - 22. März 2011 in Lübeck*, H. Handels, J. Ehrhardt, T. M. Deserno, H.-P. Meinzer, and T. Tolxdorff, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 154–158, ISBN: 978-3-642-19335-4. DOI: 10.1007/978-3-642-19335-4_33. [Online]. Available: https://doi.org/10.1007/978-3-642-19335-4_33.

[100] S. Bauer, J. Wasza, S. Haase, N. Marosi, and J. Hornegger, "Multi-modal surface registration for markerless initial patient setup in radiation therapy using microsoft's Kinect sensor," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1175–1181, 2011. DOI: 10.1109/ICCVW.2011.6130383.

[101] V. Armenise, "Estimation of the 3d pose of objects in a scene captured with kinect camera using cad models," Master's thesis, KTH Royal Institute of Technology, Computer Science, 2013, p. 79.

[102] G. Yu, M. Liu, T. Liu, and L. Guo, "Estimation of point cloud object pose using particle swarm optimization," in *Proceedings of the International Conference on Machine Vision and Applications*, 2018, pp. 1–7. DOI: 10.1145/3220511.3220512.

[103] Y. Wang, H. Xu, G. Yu, and J. Liang, "Coarse Pose Registration Using Local Geometric Features," *International Symposium on Computer Science and Artificial Intelligence(ISCSAI)*, vol. 1, pp. 45–48, 2017. DOI: 10.26480/iscsai.01.2017.45.48.

[104] J. G. Dos Santos and J. P. S. Do Monte Lima, "3D object tracking in RGB-D images using particle swarm optimization," *Proceedings - 19th Symposium on Virtual and Augmented Reality, SVR 2017*, vol. 2017-November, pp. 107–115, 2017. DOI: 10.1109/SVR.2017.22.

[105] C. Kim, J. Lee, M. Cho, and C. Kim, "Fully automated registration of 3D CAD model with point cloud from construction site," *Proceedings of the 28th International Symposium on Automation and Robotics in Construction, ISARC 2011*, pp. 917–922, 2011. DOI: 10.22260/isarc2011/0169.

[106] A. W. Fitzgibbon, “Robust registration of 2d and 3d point sets,” *Image and Vision Computing*, vol. 21, no. 13, pp. 1145–1153, 2003, British Machine Vision Computing 2001, ISSN: 0262-8856. DOI: https://doi.org/10.1016/j.imavis.2003.09.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0262885603001835.

[107] E. Héry, P. Xu, and P. Bonnifait, “Lidar based relative pose and covariance estimation for communicating vehicles exchanging a polygonal model of their shape,” in *10th Workshop on Planning, Perception and Navigation for Intelligent Vehicles (PPNIV-IROS 2018)*, 2018.

[108] ——, “Pose and covariance matrix propagation issues in cooperative localization with lidar perception,” in *2019 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2019, pp. 1219–1224.

[109] T. Sandy, M. Giftthaler, K. Dörfler, M. Kohler, and J. Buchli, “Autonomous repositioning and localization of an in situ fabricator,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 2852–2858.

[110] A. Gawel, H. Blum, J. Pankert, K. Krämer, L. Bartolomei, S. Ercan, F. Farshidian, M. Chli, F. Gramazio, R. Siegwart, *et al.*, “A fully-integrated sensing and control system for high-accuracy mobile robotic building construction,” in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2019, pp. 2300–2307.

[111] R. Kümmerle, B. Steder, C. Dornhege, A. Kleiner, G. Grisetti, and W. Burgard, “Large scale graph-based SLAM using aerial images as prior information,” *Autonomous Robots*, vol. 30, no. 1, pp. 25–39, 2011, ISSN: 09295593. DOI: 10.1007/s10514-010-9204-1.

[112] J. Canny, “A Computational Approach to Edge Detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 6, pp. 679–698, 1986, ISSN: 01628828. DOI: 10.1109/TPAMI.1986.4767851.

[113] X. Liu, L. Zhang, S. Qin, D. Tian, S. Ouyang, and C. Chen, “Optimized loam using ground plane constraints and segmatch-based loop detection,” *Sensors (Switzerland)*, vol. 19, no. 24, 2019, ISSN: 14248220. DOI: 10.3390/s19245419.

[114] J. Behley and C. Stachniss, “Efficient surfel-based slam using 3d laser range data in urban environments.,” in *Robotics: Science and Systems*, 2018, p. 59.

[115] X. Chen, A. Milioto, E. Palazzolo, P. Giguere, J. Behley, and C. Stachniss, “SuMa++: Efficient LiDAR-based Semantic SLAM,” *IEEE International Conference on Intelligent Robots and Systems*, pp. 4530–4537, 2019, ISSN: 21530866. DOI: 10.1109/IROS40897.2019.8967704.

[116] K. Ćwian, M. R. Nowicki, J. Wietrzykowski, and P. Skrzypczyński, “Large-scale lidar slam with factor graph optimization on high-level geometric features,” *Sensors*, vol. 21, no. 10, 2021, ISSN: 14248220. DOI: 10.3390/s21103445.

[117] Z. Chai, X. Shi, Y. Zhou, and Z. Xiong, “A real-time global re-localization framework for 3D LiDAR SLAM,” pp. 1–7, 2021. arXiv: 2109.00200. [Online]. Available: http://arxiv.org/abs/2109.00200.

[118] X. Chen, T. Läbe, A. Milioto, T. Röhling, J. Behley, and C. Stachniss, “Overlapnet: A siamese network for computing lidar scan similarity with applications to loop closing and localization,” *Autonomous Robots*, vol. 46, no. 1, pp. 61–81, 2022.

[119] R. Muñoz-Salinas, M. J. Marín-Jimenez, and R. Medina-Carnicer, “SPM-SLAM: Simultaneous localization and mapping with squared planar markers,” *Pattern Recognition*, vol. 86, pp. 156–171, 2019, ISSN: 00313203. DOI: 10.1016/j.patcog.2018.09.003.

[120] B. Pfrommer and K. Daniilidis, “TagSLAM: Robust SLAM with Fiducial Markers,” 2019. arXiv: 1910.00679. [Online]. Available: http://arxiv.org/abs/1910.00679.

[121] J. Wang, G. Zhang, and Z. You, “Design rules for dense and rapid Lissajous scanning,” *Microsystems and Nanoengineering*, vol. 6, no. 1, pp. 4–10, 2020, ISSN: 20557434. DOI: 10.1038/s41378-020-00211-4. [Online]. Available: http://dx.doi.org/10.1038/s41378-020-00211-4.

[122] J. W. Anderson and G. M. Clayton, “Lissajous-like scan pattern for a gimballed LIDAR,” *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM*, vol. 1, pp. 1171–1176, 2014. DOI: 10.1109/AIM.2014.6878240.

[123] S. Agarwal, K. Mierle, and T. C. S. Team, *Ceres Solver*, version 2.1, Mar. 2022. [Online]. Available: https://github.com/ceres-solver/ceres-solver.

List of Figures

List of Tables