Chair of Computational Modeling and Simulation
Department of Civil, Geo and Environmental Engineering
Technical University of Munich

Model Aware LiDAR Odometry and Mapping (MA-LOAM): Improving Simultaneous Localization and Mapping accuracy by robustly leveraging a Building Information Model

Scientific work to obtain the degree

Master of Science (M.Sc.)

at the Department of Civil, Geo and Environmental Engineering of the Technical University of Munich.

Supervised by Prof. Dr.-Ing. André Borrmann
M. Sc. Miguel Arturo Vega Torres
Chair of Computational Modeling and Simulation

M. Sc. Martin Oelsch
Chair of Media Technology

Submitted by Muhammad Hamza Sattar [redacted]

Submitted on 12th February 2022

Abstract

The ability to accurately determine the position of an agent in an unknown environment is of crucial importance for the Simultaneous Localization And Mapping (SLAM) problem. Better localization and odometry estimates lead to improved map generation. This map forms the basis of most robot navigation stacks and is essential for driving without human interference. With the increase in automation along with the use of robotics, there is an increased need to refine the SLAM related methodologies. While indoor navigation usually lacks access to reliable Global Positioning System (GPS) data, it has the advantage of having a known building model in most cases. This extra level of information can be exploited to boost the quality of both the localization and mapping modules. As the real environment usually differs significantly from the theoretical model, a methodology that is resilient to outliers and scan model differences is proposed. This approach is added on top of Lidar Odometry And Mapping (LOAM) and also made open source to the public.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

3D	3-Dimensional
AABB	Axis Alligned Bounding Box
B-Rep	Boundary Representation
BIM	Building Information Modeling
BVH	Bounding Volume Hierarchies
BVH	Bounding Volume Hierarchy
CAD	Computer Aided Design
CNN	Convolutional Neural Network
DCP	Deep Closest Point
DGR	Deep Global Registration
EKF	Extended Kalman Filter
FCGF	Fully Convolutional Geometric Features
FGR	Fast Global Registration
FLANN	Fast Library for Approximate Nearest Neighbors
FPFH	Fast Point Feature Histogram
GPS	Global Positioning System
ICP	Iterative Closest Point
IDC	Iterative Dual Correspondence
IMU	Inertial Measurement Unit
IPS	Indoor Positioning System
K-NN	K-Nearest Neighbors
KF	Kalman Filter
KITTI	Karlsruhe Institute of Technology and Toyota Technological Institute
LiDAR	Light Detection And Ranging
LOAM	Lidar Odometry And Mapping
MA-LOAM	Model Aware Lidar Odometry And Mapping
PCA	Principal Component Analysis
RANSAC	Random Sample Consensus
ROS	Robot Operating System
SEIF	Sparse Extended Information Filter
SIFT	Scale Invariant Feature Transform
SLAM	Simultaneous Localization And Mapping
SURF	Speeded Up Robust Features
SVD	Singular Value Decomposition
V-LOAM	Visual Lidar Odometry And Mapping
Wi-Fi	Wireless Fidelity

Chapter 1

Introduction

With the ever-increasing demand for automation, robots have become quite common over time. Some are relatively static such as robotic arms where they are free to move, but the base remains in almost the same position. On the contrary, some robots need to navigate actively around in an unknown environment. This navigation requires a solution to the so-called SLAM problem. It is only possible if the robot can perceive its surroundings using sensors that can either be active (e.g., Light Detection And Ranging (LiDAR), radar) or passive (e.g., Inertial Measurement Unit (IMU), camera).

Navigation may be indoor, outdoor, or hybrid, depending on the environment. The type of navigation environment determines/limits the type of sensors used. For example, for outdoor navigation, GPS may be a viable option, but for indoor navigation, it may not be feasible as GPS requires direct sight to at least four satellites, three for determining the 3-Dimensional (3D) position and one for correcting the satellite time. Obstructions may lead to degradation of the localization accuracy at best and complete failure at worst. There are various Indoor Positioning System (IPS) alternatives to overcome this limitation that rely on radio signals such as from BlueTooth or Wireless Fidelity (Wi-Fi). Other modes include acoustic, light and magnetic field signals [Curran et al., 2011, Lopez-de-Teruel et al., 2017] to name a few. The downside is that this requires additional strategically placed sensors, which can drive up the cost of the navigation stack. In certain cases, they might be unavoidable, but in the case of most modern architecture, where Building Information Modeling (BIM) is available, the environment model can either supplement or completely replace the IPS depending upon the localization accuracy requirements. The main advantage is that it leverages existing information and requires no additional sensors. This thesis mainly aims at developing a robust methodology that utilizes BIM model to improve SLAM accuracy even in cases where there are significant differences between the LiDAR scan and the reference model.

1.1 Simultaneous Localization And Mapping

Simultaneous Localization And Mapping (SLAM), first coined by [H. F. Durrant-Whyte et al., 1996], is a well studied[Parker, 2000, H. Durrant-Whyte and Bailey, 2006, Dorigo et al., 2014] problem, albeit a difficult one. It is the process of concurrently determining the position of an agent in an unknown environment while creating/updating its map given some data from sensors. Solving the SLAM problem is a highly non-linear problem that appears to be circular. If the map of an environment is known, localizing is straightforward. Similarly, if the exact position of the agent is known, creating a high fidelity

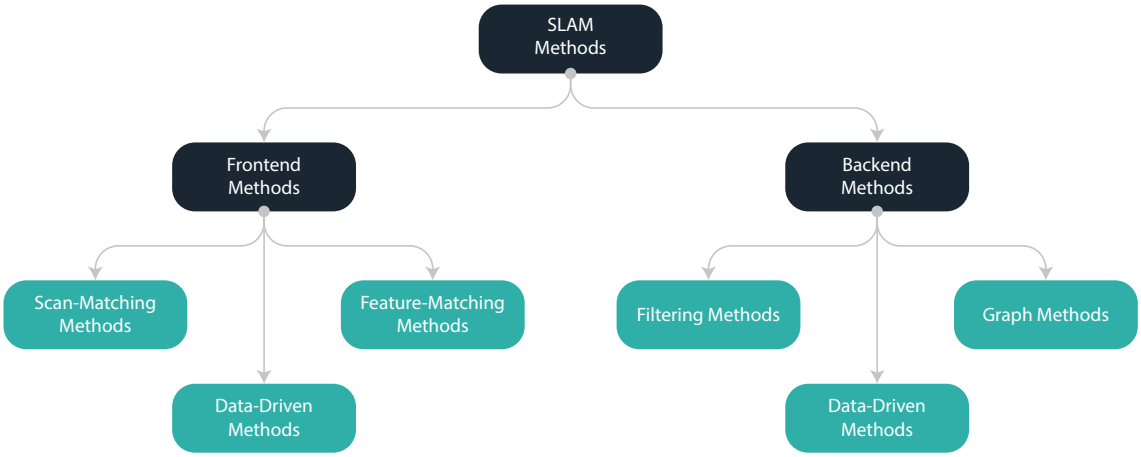


Figure 1.1: SLAM classification based on software architecture [KUZMIN, 2018]

map is also quite facile. In reality, neither the map nor the exact position of the agent is known. Nonetheless, there exist many methodologies to approximately solve both at the same time. They can generally be classified as shown in fig. 1.1 depending on how these techniques approach the problem:

- **Backend methods:** These methods perform the SLAM computation when all the data for sensors has been collected. They are more resilient to errors and depend on probabilistic models to correct bad observations. Though they can work independently, they are usually accompanied by frontend methods.
    - **Filtering methods:** One of the oldest approaches for tackling the SLAM problem. Some notable examples are Extended Kalman Filter (EKF) [JULIER and UHLMANN, 1997] which is the extension of Kalman Filter (KF) for non-linear problems, particle methods [GRISETTI et al., 2007] and Sparse Extended Information Filter (SEIF) [EUSTICE et al., 2005] to name a few. These approaches are rarely used nowadays apart from particle filters.
    - **Graph methods:** As the name suggest, the observations are stored in form of a graph [THRUN and MONTEMERLO, 2006] which is later optimized to compute the transversed path. Additional constraints such as loop closure [LATIF et al., 2013] can greatly reduce the errors in the final solution.
    - **Data driven methods:** These methods depend mainly upon advances in machine learning and neural networks. They can either aid classical techniques like EKF [CHOI et al., 2007] and graph SLAM [NASEER et al., 2015] or be completely independent [TATENO et al., 2017].
- **Frontend methods:** These methods mainly depend on energy / error minimization algorithms. If the underlying data comes from highly precise sensors, they can be used independently. Otherwise, they are paired up with backend methods where they act as a prior.

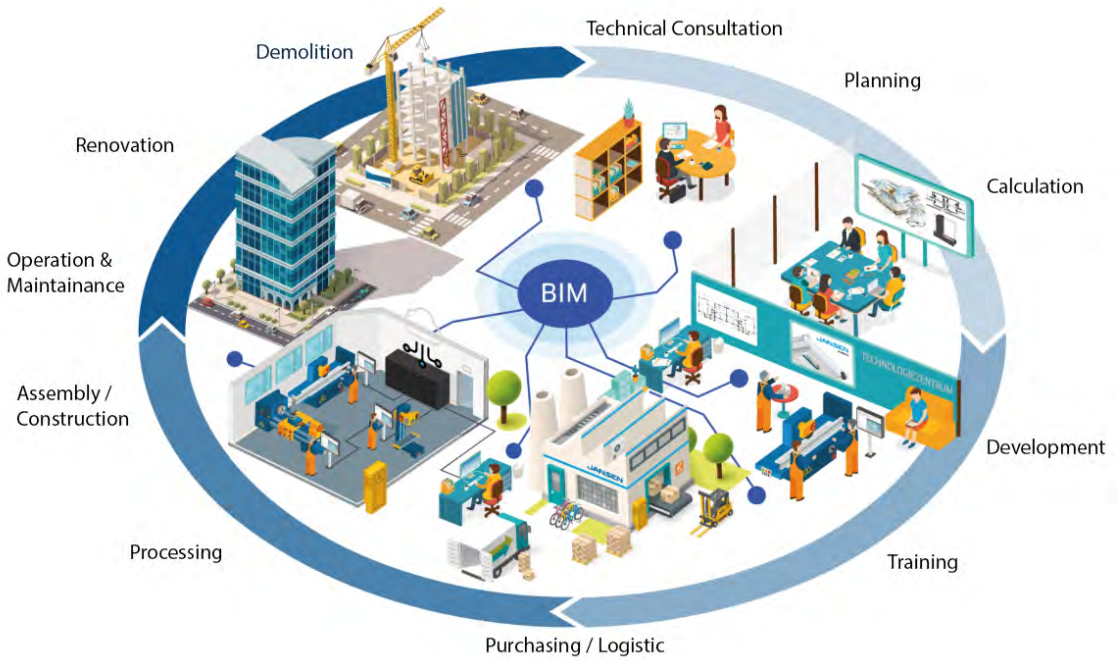


Figure 1.2: BIM lifecycle [“Planung: BIM-Modelle von Jansen Stahlsystemen”, 2021]

- **Scan matching methods**: They operate by matching subsequent scans coming from the sensors such as LiDAR. Some examples are ICP [Arun et al., 1987] and Iterative Dual Correspondence (IDC) [Lu and Milios, 1997].
- **Feature matching methods:** Unlike scan matching algorithms, these methods do not try to match all the scan data but rather extract distinct features from the environment and match those. Some examples are Scale Invariant Feature Transform (SIFT) [Lowe, 2004] and Speeded Up Robust Features (SURF) [Bay et al., 2006].
- **Data driven methods:** Like backend methods, machine learning is also being used for frontend methods [Wang and Solomon, 2019, H. Yang et al., 2020, C. Chen et al., 2020].

1.2 Lidar Odometry And Mapping

Lidar Odometry And Mapping (LOAM) [J. Zhang and Singh, 2014] is state of the art algorithm that uses LiDAR sensor for solving the SLAM problem. It belongs to the feature matching methods discussed above. It computes edge and surface features for each scan and finds correspondences in subsequent scans. By performing this process iteratively, odometry is estimated, which is used in the mapping process. This thesis builds on top of this algorithm and is discussed later in more detail.

1.3 Building Information Modeling

The term Building Information Modeling (BIM) was first used in a modern sense in a 1985 paper published in the later year [RUFFLE, 1986]. Despite being explored early on, it did not become popular till years later due to restrictive computer hardware availability. Nowadays, it is a common technique for managing and optimizing construction projects. It is quite different from standard Computer Aided Design (CAD) models as it is multifaceted in nature (fig. 1.2). It stores building information hierarchically and allows for better coordination among different groups. If the BIM model also include time information, it is known as 4D BIM [HOLNESS, 2008], 3 dimensions representing the CAD model while 4_{th} dimension represents time. In such a case, BIM model can be queried for a time-dependent environment model and used in the SLAM computation rather than the finished model.

1.4 Thesis Overview

After briefly discussing SLAM, BIM and LOAM, it is now possible to present a concrete overview of this thesis. As already mentioned before, most modern buildings have a BIM model or CAD model at the very least. This thesis proposes a robust methodology for integrating the environment model into the LOAM framework.

The point cloud obtained from the LiDAR sensor is first preprocessed by clustering similar points together. For clustering, the points are placed in voxels of defined size. For each point, surface normal is computed using the neighbor's information. Then, points are clustered together within each voxel using normals and curvature as a similarity parameter. The number of clusters per voxel and minimum cluster size is also pre-defined. These parameters control the retained detail and act as the first outlier rejection step.

The points are then iteratively projected onto the environment model. This projection is the foundation of the optimization problem, which is solved later. For loss function, robust kernels [HUBER, 1992, BELAGIANNIS et al., 2015] are used instead of $l2-$loss which acts as a second outlier rejection step by reducing the impact of outliers.

The clustered point cloud is then formulated as classical ICP problem with the point-to-plane metric using the environment model as reference. Point to mesh features set is again filtered out at every iteration of ICP if the distance exceeds the set threshold acting as a third outlier rejection step.

This optimization step is then added to the LOAM problem, and the weights are set proportional to the cardinality of the edge, surface, and mesh features, respectively. The optimization problem is then solved and evaluated for improvement.

ICP is a local refinement algorithm. The optimization may get stuck in local minima if the initial guess is too far from the ground truth. Multiple global registration algorithms are tested to tackle this problem. After evaluating their performance and robustness, a recent

deep learning approach, namely Deep Global Registration (DGR) [CHOY et al., 2020], is selected for global registration.

1.5 Research Objectives

As mentioned at the start of this chapter, the main objective is to robustly integrate the BIM model into the LOAM algorithm to improve the SLAM accuracy. Specifically, the approach should deal with the following questions:

1. What useful information can be extracted from the BIM model?
2. How can the extracted information be integrated into LOAM algorithm?
3. In what ways can the effect of noise/outliers be minimized? How does it stack up against the base algorithm?
4. Is it possible to run the methodology in real-time? What part of the algorithm contributes the most to execution time?
5. Can the approach generalize to environments where the scan differs significantly from the model?

1.6 Reading Guide

The minimum theoretical background necessary for understanding this thesis is discussed in chapter 2. Literature review and related methodologies are covered in chapter 3. Chapter 4 presents the methodology developed while working on this thesis. It also includes design decisions and other details about why a specific technique was chosen. Results and comparative study with other similar algorithms are done in chapter 5. Finally, in chapter 6, a general critique is presented along with recommendations and conclusion.

Chapter 2

Theoretical Background

This chapter discusses the theoretical background necessary to understand the proposed methodology in this thesis properly. It briefly covers efficient nearest neighbors search using a k-D tree, estimation of the surface normal of a point cloud, voxelization, gradient-based optimization, automatic differentiation, ICP, robust loss functions, fast point to mesh projections using Bounding Volume Hierarchies (BVH) and neural networks.

2.1 K-Nearest Neighbors

K-Nearest Neighbors (K-NN) is a local classification and regression algorithm. It depends on a simple paradigm: the input is similar to its neighbors. Depending upon the number of neighbors k, the input is classified using the majority. Therefore, it is common to have k as an odd number to resolve ties. Nonetheless, even values of k can be used with additional metrics such as distance to break ties. For regression, simple arithmetic mean can be used. Consider the fig. 2.1. Depending upon the selection of parameter k, the unknown circular input I can be classified. So, for example, if $k = 3$, input I belongs to the star class as $2/3$ of its neighbors belong to the star class. On the contrary, if it is chosen that $k = 5$, the input I belongs to the square class($3/5$). It is important to select a reasonable value of k, as too small or too big of a value can negatively affect the classification/regression.

Sequentially searching for neighbors is of complexity $O(n)$ where n is the total number of points in the search domain. This search becomes computationally expensive for large datasets and therefore requires efficient handling. There exist many acceleration strategies to reduce the computational complexity of the nearest neighbor search. One of the most common ones is a k-D tree [BENTLEY, 1975].

2.1.1 K-D Tree

K-D trees are a direct extension of binary trees in k dimensions. It recursively partitions the search space based on the median along a particular axis and creates the binary tree. This partitioning reduces the complexity of the nearest neighbor search from $O(n)$ to amortized $O(\log n)$. Building a k-D tree also requires some additional work. If a suitable algorithm is used for computing the median [M. BLUM et al., 1973], building a k-D tree takes $O(n \log n)$.

Figure 2.2 shows exactly how the dataset is partitioned in the example presented in fig. 2.1. While a k-d tree does accelerate nearest neighbor queries, it should be remembered that

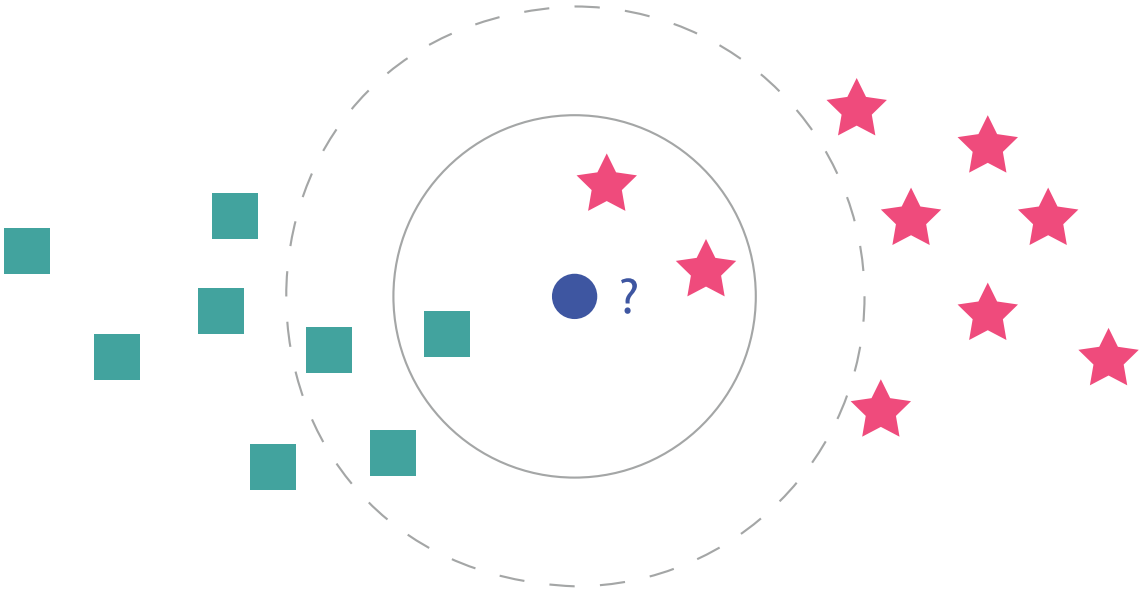


Figure 2.1: With $k = 3$, the target belongs to the star class while with $k = 5$, it belongs to the square class. In case of regression, the neighbors property are averaged (adapted from AJANKI, 2007)

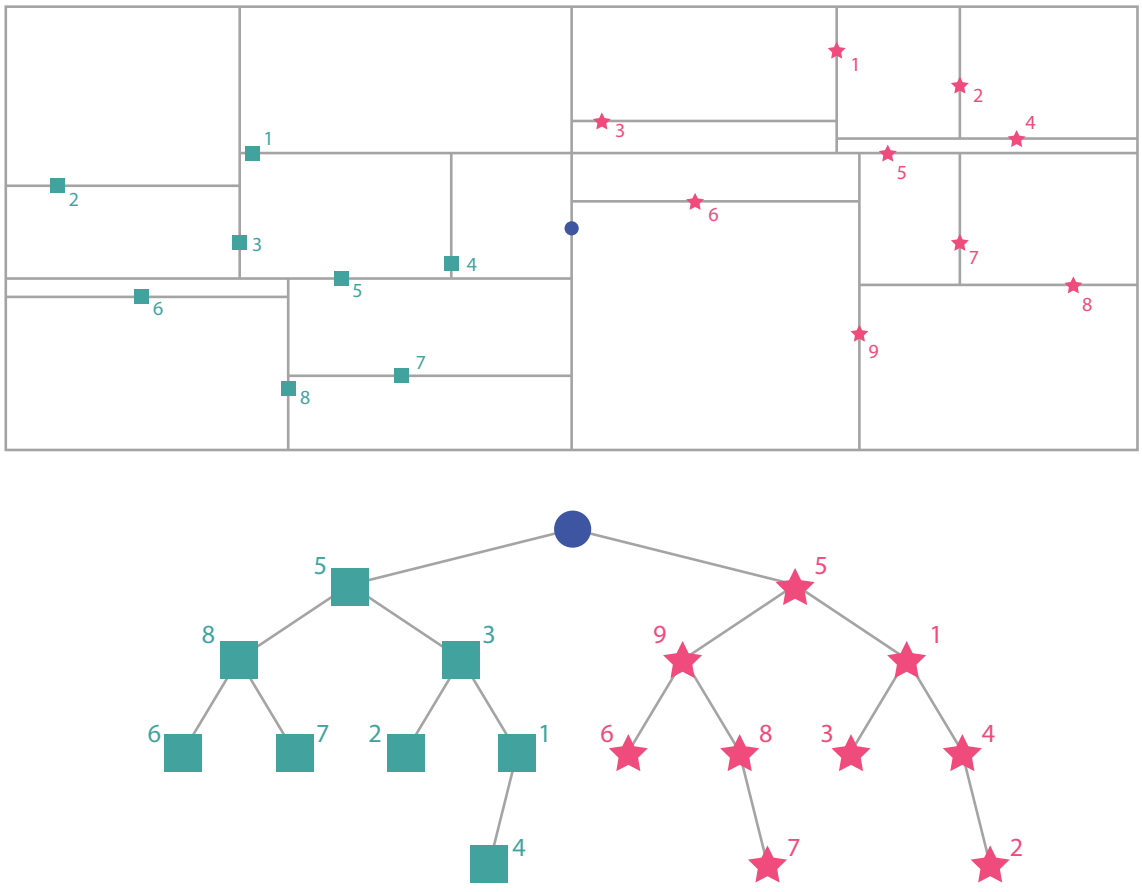


Figure 2.2: Top: K-D tree partitioning with x as starting medial axis; Bottom: k-D tree after partitioning

Algorithm 2.1: Normal Estimation

```
Inputs: P = points, k = number of neighbors
Output: Normals
begin
    normals = {}
    for p in P:
        neighbors = knn_kdtree(p, P, k)
        p̄ = sum(neighbors) / k

        // Compute the covariance matrix
        p̂ = neighbors − p̄
        cov = (p̂ * p̂.transpose()) / k

        // Compute the eigen values using singular
        // value decomposition
        U, V = svd(cov)

        // Smallest eigen value corresponds to
        // the normal vector
        normals.push(min(U))
    end
    return normals
end
```

they suffer from the curse of dimensionality. As the number of dimensions increases, it becomes harder to rule out possible matches, which leads to a degradation in performance. There have been multiple efforts [INDYK and MOTWANI, 1998 BERCHTOLD et al., 1998 KUO and SLOAN, 2005] to resolve the issue some of which relax the condition of exact neighbors in favor of approximate ones. One of the most prominent algorithms for approximate neighbors search is Fast Library for Approximate Nearest Neighbors (FLANN) [MUJA and LOWE, 2009] which is extensively used in this thesis.

2.2 Normal Estimation

There are many ways to compute the surface normals given a point cloud. One of the simplest and popular one is using Principal Component Analysis (PCA) for computing the covariance matrix of each point by considering k neighbors. It can be computed using the following equation:

$$\mathcal{C} = \frac{1}{k} \sum_{i=1}^{k} (p_i - \overline{p}) \cdot (p_i - \overline{p})^T \tag{2.1}$$

where k is number of neighbors, p_i is the point under consideration and $\overline{p}$ is the mean of the k neighbors. The covariance matrix can then be decomposed using eigenvalue decomposition to get the three eigenvectors.

$$\mathcal{C} \cdot \overrightarrow{v_j} = \lambda_j \cdot \overrightarrow{v_j} \quad j \in \{0, 1, 2\} \tag{2.2}$$

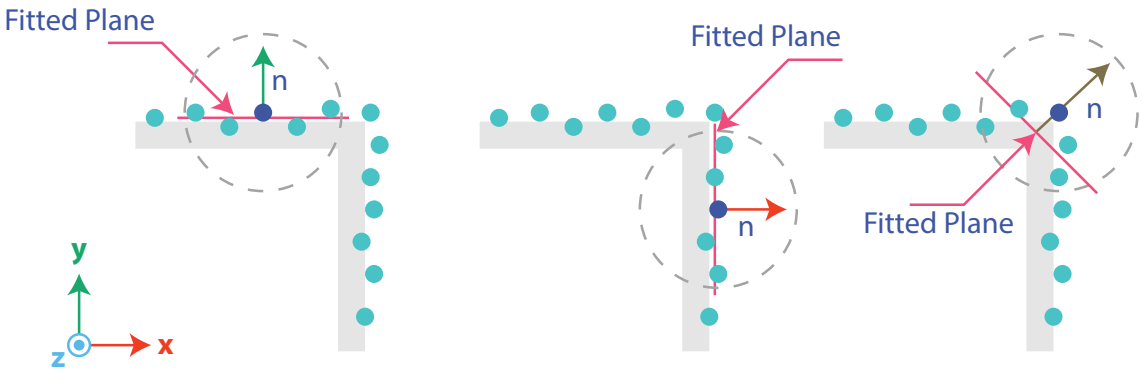


Figure 2.3: Three different scenarios depicting normal computation with $k = 4$

The smallest eigenvalue represents the normal vector. The whole procedure is summarized in Algorithm 2.1

The normals obtained using Singular Value Decomposition (SVD) may not have a consistent sign. A normal vector in the negative direction can also be a valid candidate. To ensure that all the signs are consistent, a post-processing step is necessary, which considers the origin of the point cloud. The curvature σ of each point in the point cloud can also be computed using the already calculated eigenvectors λ:

$$\sigma = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \tag{2.3}$$

Where λ_0 represents the smallest eigenvector, λ_1 and λ_2 are eigenvectors along the fitted plane. A 2D view of a 3D wall is shown in fig. 2.3 where normals are computed using the mentioned technique.

2.3 Voxelization

Voxelization is the process of storing data in a discrete grid. Each point that lies within the voxel is grouped together. The shape of the grid can vary. Two of the most common branches are:

- **Cartesian grid:** The voxels are represented by a cartesian grid of uniform size, i.e., all the cells have the same size. The empty voxels may be removed from the transversal path. Figure 2.4 shows voxelization in 2D using a uniform cartesian grid
- **Quadtree:** Rather than using equally sized voxels, it is possible to recursively subdivide the parent voxel into four equal parts [SAMET, 1984]. This process is applied recursively until each child voxel contains only one point or if the maximum depth d is reached (fig. 2.5).

For the cartesian grids, the controlling parameter is the cell size, whereas, for the quadtree, it is the tree depth. Both voxelization techniques are easily extendable into 3D, in which

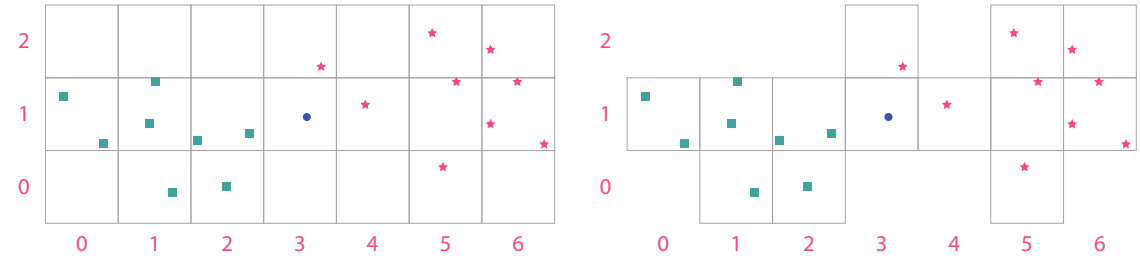


Figure 2.4: Left: Voxelization using cartesian grid; Right: Empty voxels removed

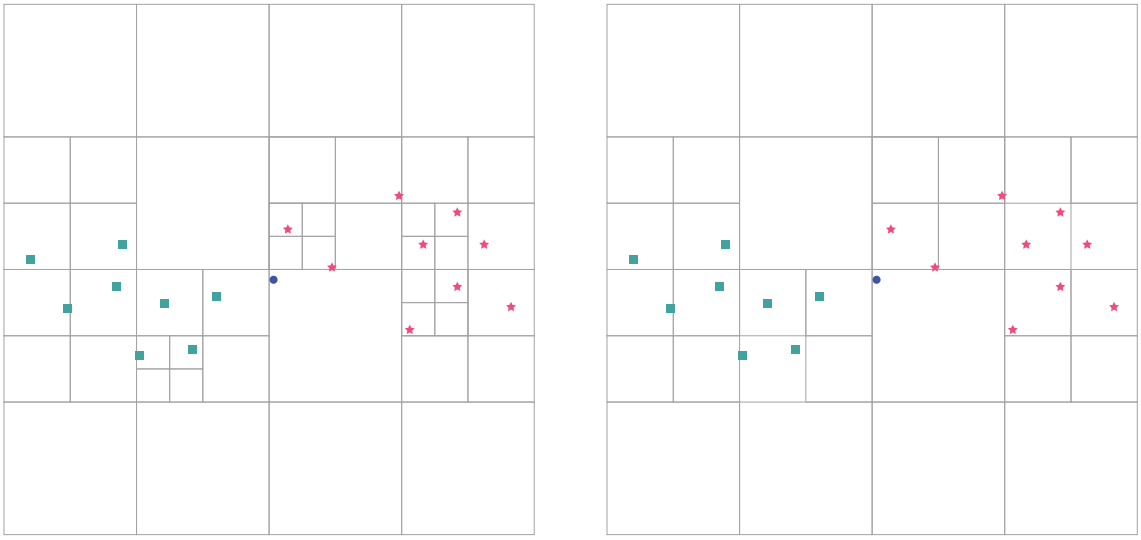


Figure 2.5: Quad tree voxelization with: Left: $d = 4$, each voxel contains a single element; Right: $d = 3$, maximum depth is reached

case, the cell becomes a cuboid. The 3D variant of quadtree is known as octree [MEAGHER, 1982]. As the name suggests, quadtree and octree are represented by tree graphs, with each non-leaf node containing 4 and 8 children, respectively. One of the common uses of voxelization is as a down-sampling filter where the points within the same voxel are merged and represented by a single point at their centroid. There exist more intelligent alternatives for merging the points.

2.4 Gradient Based Optimization

Optimization is one of the most frequently encountered problems in engineering. It involves refining design variables x based on some criteria such that the objective function $f(x)$ is minimized. Maximization problem are posed as $g(x) = -f(x)$. If the objective function is continuous and differentiable, gradient-based methods are preferred. Convex problems are quite easy to solve and guarantee convergence to global optima. In reality, it is rarely the case, and therefore, for non-convex, finding global optima is not always possible. One of the most frequent classes of such problems is minimizing the sum of squared error which is encountered everywhere, from simple curve fitting to robotics to machine learning. Some of the commonly used gradient-based algorithms are briefly discussed below. Note that they are all dependent on the quality of the initial guess.

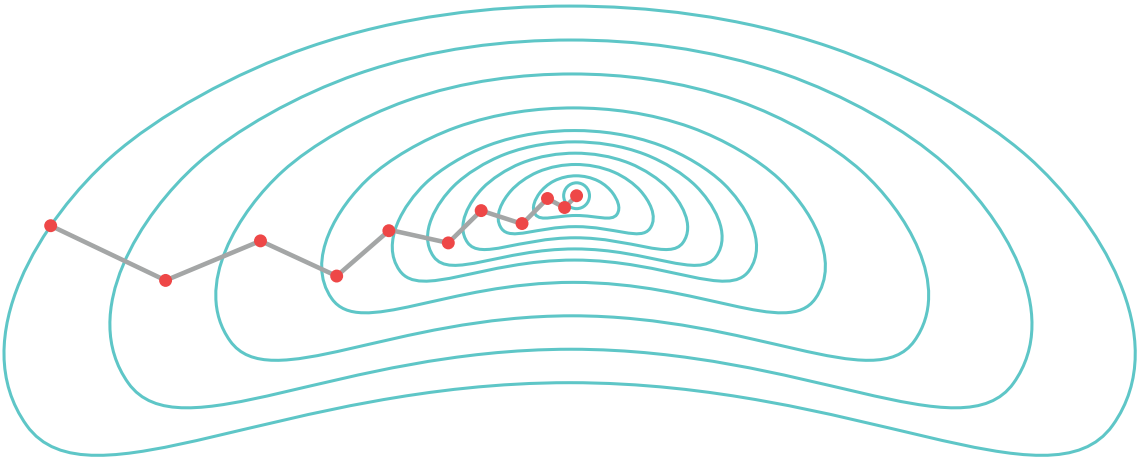


Figure 2.6: Oscillatory behavior of gradient descent algorithm (adapted from LEÓN, 2021)

2.4.1 Gradient descent

It is one of the most basic approaches to solve the multi-variable optimization problem [HADAMARD, 1908]. The general idea is relatively straightforward: iteratively step in the opposite direction of the gradient. As gradient points towards the local ascent direction, moving in the other direction should lead towards the minima, thus the name gradient descent. One of the main problems of this algorithm is that it can easily get stuck in local minima. Even for quadratic problems, it tends to oscillate a lot, as shown in fig. 2.6. Mathematically, the iterative update procedure is given by:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F\left(\mathbf{x}_n\right), n \geq 0 \tag{2.4}$$

where $\mathbf{x}_n$ is the current value of the design variables, $\mathbf{x}_{n+1}$ is the next estimate, γ_n is the learning rate at each iteration and $\nabla F\left(\mathbf{x}_n\right)$ is the Jacobian at the current point. The learning rate γ can be static as well as dynamic. For example, it can be computed using line search or trust-region methods [BARZILAI and BORWEIN, 1988]:

$$\gamma_n = \frac{\left|\left(\mathbf{x}_n - \mathbf{x}_{n-1}\right)^T \left[\nabla F\left(\mathbf{x}_n\right) - \nabla F\left(\mathbf{x}_{n-1}\right)\right]\right|}{\left\|\nabla F\left(\mathbf{x}_n\right) - \nabla F\left(\mathbf{x}_{n-1}\right)\right\|^2} \tag{2.5}$$

The convergence rate of gradient descent is linear.

2.4.2 Newtons Method

Newtons methods is another method for optimizing a multi-variable objective function. For finding the solution, it not only utilizes the Jacobian matrix but also the Hessian matrix (second derivatives). The Hessian matrix represent the curvature information. The iterative approach to updating the design variables is given by:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \frac{\nabla F\left(\mathbf{x}_n\right)}{\nabla^2 F\left(\mathbf{x}_n\right)}, n \geq 0 \tag{2.6}$$

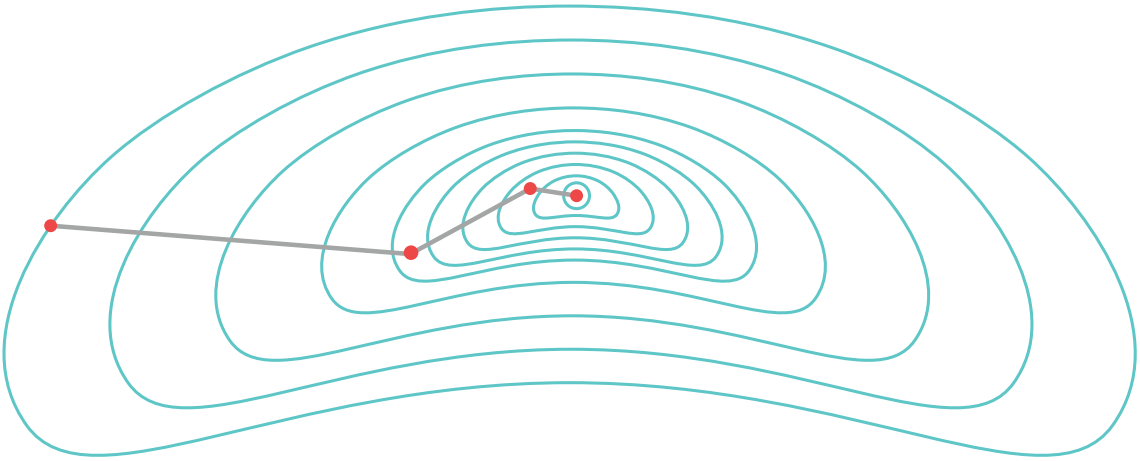


Figure 2.7: Faster convergence with Newtons method (adapted from LEÓN, 2021)

where $\nabla^2 F(\mathbf{x}_n)$ is the Hessian matrix at the current point. The rest of the entities are same as that of gradient descent (eq. (2.4)).

The convergence rate of Newton's method is quadratic, though it is also more expensive due to the computation of the Hessian matrix.

2.4.3 Levenberg–Marquardt algorithm

The earlier two methods discussed are used for general optimization. In a case where the underlying objective function represents a non-linear least square problem, Levenberg–Marquardt algorithm [MORÉ, 1978] is preferable as it is more robust and faster. It is a combination of gradient descent and Newton's method. The parameter update equation is as follows:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \frac{\nabla F(\mathbf{x}_n)}{\nabla^2 F(\mathbf{x}_n) + \lambda I}, n \geq 0, \lambda in 0, \inf \tag{2.7}$$

Where λ is the damping term, when λ is 0, the problem becomes equivalent to Newton's method. On the contrary, as λ increases, the problem becomes closer and closer to gradient descent. As the form of the objective function is known (sum of squared errors), it is possible to approximate the Hessian matrix rather than computing it exactly.

2.5 Automatic Differentiation

The optimization methods discussed above are dependent on the computation of gradients. It is possible to evaluate the analytical derivatives but the main problem is that it is only limited to simpler functions and requires hand calculations. An alternative to this is numerical differentiation, the simplest of which is of the form:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \tag{2.8}$$

As $\lim_{h\to 0}$, the approximation reaches the true value. The main problem with this approach is that the quality of the gradient depends upon the value of h as it can never be made smaller than machine precision. At first, setting h to the smallest possible value might seem reasonable, but in fact, it also degrades the accuracy as machines have finite precision and truncate mantissa whenever it exceeds the underlying type limits. For example, under IEEE standard 754 [KAHAN, 1996], 64 bit floats are represented as represented in table 2.1. Due to this finite memory conundrum, h somewhere in-between gives the best possible accuracy and is problem-dependent. Another downside of numerical differentiation is that it scales badly with the number of dimensions ($O(n)$) and becomes computationally expensive.

Table 2.1: IEEE-754 float 64 representation

	Bits
Sign	1
Exponent	11
Mantissa	53 [a]

[a] 52 bits are stored explicitly. In binary, the leading digit can either be 0 or 1. By deciding to not store leading 0 (which makes lead digit always 1), 53 bits can be saved in place of 52 bits

To overcome the limitations of both the aforementioned methods, another technique exists, namely automatic differentiation, which computes exact derivatives, is cheap to evaluate, and does not suffer from numerical issues. It requires more involved implementation but is well worth the effort. It is based on dual numbers and primarily has two modes: forward and reverse. Both of these modes depend mainly on the chain rule of differentiation.

2.5.1 Dual Numbers

Dual numbers are an extension of real numbers into a hyper-complex plane. They are represented as follows:

$$D = a + b\epsilon \tag{2.9}$$

with property ϵ is really small such that $\epsilon^2 = 0$. Addition and subtraction are pretty straightforward and done component wise. Multiplication and division are commutative. Multiplication between 2 dual numbers D_0 and D_1 is as follows:

$$\begin{aligned} D_0 * D_1 &= (a_0 + b_0\epsilon)(a_1 + b_1\epsilon) && (2.10)\\ &= a_0a_1 + (a_0b_1 + a_1b_0)\epsilon + b_0b_1\epsilon^2 && (2.11)\\ &= a_0a_1 + (a_0b_1 + a_1b_0)\epsilon && (2.12) \end{aligned}$$

Similarly, division can be carried out by multiplying and dividing the fraction by the denominator conjugate value. Using the duality property, exact derivatives of analytical functions

can be computed using the Taylor series approximation. Consider the Taylor series of a general function $f(a)$ around the point $b\epsilon$:

$$f(a+b\epsilon) = f(a) + \frac{f'(a)}{1!}(b\epsilon) + \frac{f''(a)}{2!}(b\epsilon)^2 + \frac{f'''(a)}{3!}(b\epsilon)^3 + \cdots \tag{2.13}$$

Once again, using the property $\epsilon^2 = 0$, all higher-order terms cancel out.

$$f(a+b\epsilon) = f(a) + b\epsilon f'(a) \tag{2.14}$$

Therefore, in order to evaluate any function and its derivative at a, switching to dual number arithmetic and evaluating at $a + \epsilon\,(b = 1)$ instead leads to the real part representing $f(a)$ while the dual part representing $f'(a)$ as per eq. (2.14).

2.5.2 Examples

Consider the following two examples:

- With $f(x) = x^3 + 1$, substituting in $x + \epsilon$ leads to:

$$\begin{aligned} f(x) &= x^3 + 1 \\ f(x+\epsilon) &= (x+\epsilon)^3 + 1 \\ &= x^3 + 3x^2\epsilon + 3x\epsilon^2 + \epsilon^3 + 1 \\ &= x^3 + 1 + \left(3x^2\right)\epsilon \end{aligned}$$

  Indeed the real part $x^3 + 1$ represents the initial function while the dual part $3x^2$ represents its derivative.

- A slightly more involved example is that of $f(x) = \sin x$:

$$\begin{aligned} f(x) &= \sin x \\ &= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots && \text{Taylor series expansion} \\ f(x+\epsilon) &= x + \epsilon - \frac{(x+\epsilon)^3}{3!} + \frac{(x+\epsilon)^5}{5!} - \frac{(x+\epsilon)^7}{7!} + \cdots \\ &= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots + \left(1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots\right)\epsilon \\ &= \sin x + (\cos x)\,\epsilon \end{aligned}$$

  All other trigonometric functions have a similar procedure to compute the automatic derivatives.

In general, the methodology in the last example holds up for most analytical functions. With the ability to compute derivatives when the function is evaluated, it is now possible to integrate it with the gradient-based optimization technique discussed in section 2.4.

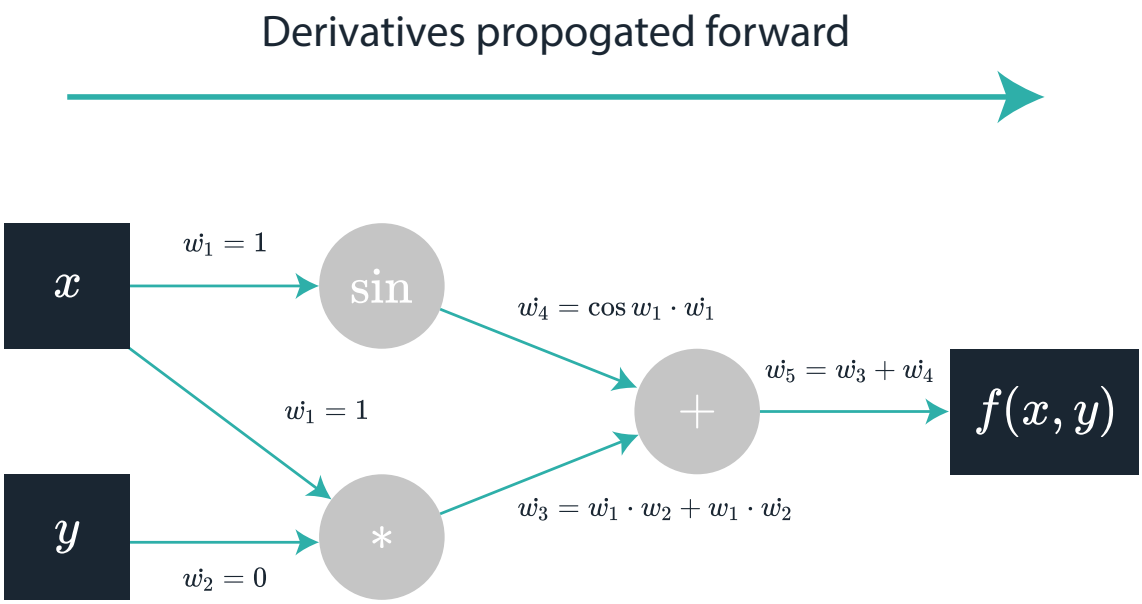


Figure 2.8: Computational graph of automatic differentiation with respect to x in forward mode for eq. (2.15) with $w_1 = x$, $w_2 = y$ and $\dot{w}_5 = \partial f / \partial x$ (adapted from BERLAND, 2007a)

2.5.3 Forward Mode

In the forward mode, the independent variables are fixed and propagated forward. Consider the following function:

$$f(x, y) = xy + \sin x \tag{2.15}$$

Here, the independent variable are x and y. Recursive substitution leads to the following:

$$\begin{aligned} z &= f(x, y) \\ &= xy + \sin x \\ &= w_1 w_2 + \sin w_1 \\ &= w_3 + w_4 \\ &= w_5 \end{aligned} \tag{2.16}$$

So, for example, if the derivative of $f(x, y)$ with respect to x is required, following steps will be performed (as per eq. (2.16)).

Table 2.2: Forward mode decomposition of eq. (2.15) with respect to x

Substitutions	Derivatives $\partial / \partial x$	Derivatives simplified
$w_1 = x$	$\dot{w}_1 = 1$	1
$w_2 = y$	$\dot{w}_2 = 0$	0
$w_3 = w_1 \cdot w_2$	$\dot{w}_3 = \dot{w}_1 \cdot w_2 + w_1 \cdot \dot{w}_2$	y
$w_4 = \sin w_1$	$\dot{w}_4 = \cos w_1 \cdot \dot{w}_1$	$\cos x$
$w_5 = w_3 + w_4$	$\dot{w}_5 = \dot{w}_3 + \dot{w}_4$	$y + \cos x$

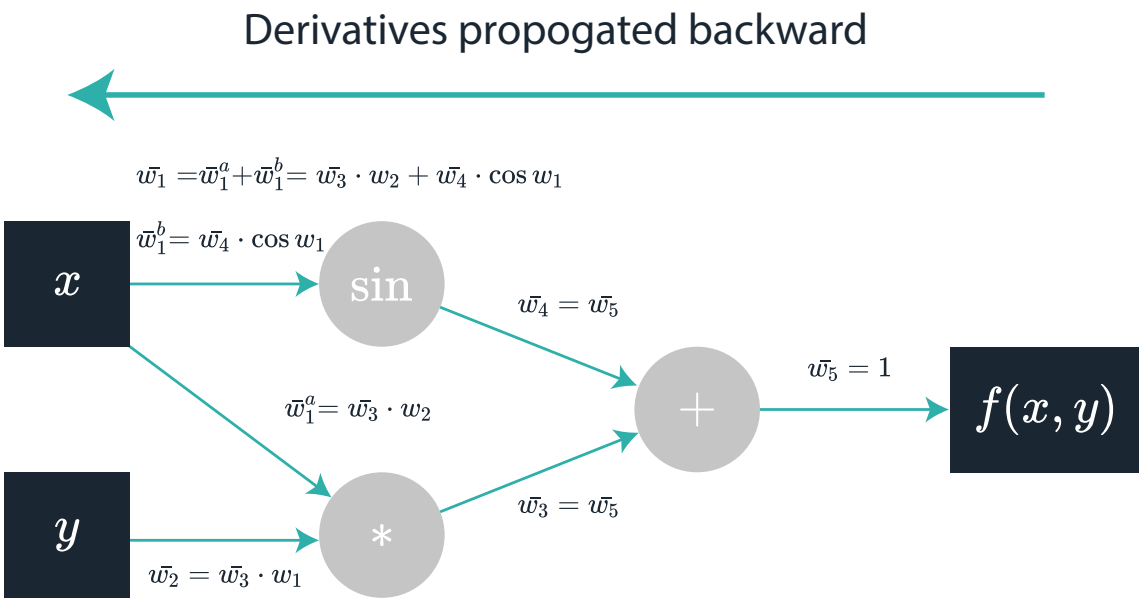


Figure 2.9: Computational graph of automatic differentiation in reverse mode for eq. (2.15) with $w_1 = x$, $w_2 = y$, $\bar{w_1} = \partial f / \partial x$ and $\bar{w_2} = \partial f / \partial y$ (adapted from BERLAND, 2007b)

As $w_5 = z = f(x, y)$, the derivative $\partial f / \partial x$ is equal to $\dot{w_5}$. For forward mode, the time complex is $O(n_{input})$ and requires constant space.

2.5.4 Reverse Mode

Reverse mode is a bit more complicated. It relies heavily on the multi-variable chain rule. The dependent variables are fixed and propagated in the reverse direction. The main advantage is that it is only dependent on the number of outputs, therefore a single backward pass computes derivatives in case of eq. (2.15) results in both $\partial f / \partial x$ and $\partial f / \partial y$.

Table 2.3: Reverse mode decomposition of eq. (2.15) with w reused from table 2.2

Substitutions	Derivatives	Derivatives simplified
$\bar{w_5} = \partial w_5 / \partial w_5$	1	1
$\bar{w_4} = \bar{w_5} \cdot \partial w_5 / \partial w_4$	$\bar{w_5}$	1
$\bar{w_3} = \bar{w_5} \cdot \partial w_5 / \partial w_3$	$\bar{w_5}$	1
$\bar{w_2} = \bar{w_3} \cdot \partial w_3 / \partial w_2$	$\bar{w_3} \cdot w_1$	x
$\bar{w_1} = \bar{w_3} \cdot \partial w_3 / \partial w_1 + \bar{w_4} \cdot \partial w_4 / \partial w_1$	$\bar{w_3} \cdot w_2 + \bar{w_4} \cdot \cos w_1$	$y + \cos x$

As w_1 and w_2 correspond to x and y, therefore $\partial f / \partial x$ and $\partial f / \partial y$ are equal to $\bar{w_1}$ and $\bar{w_2}$ respectively. In a single pass, both of the derivatives are obtained. It is therefore of complexity $O(n_{output})$ and also requires additional space of $O(n_{operations})$ to store the intermediate results. Many derivatives often share almost similar interior paths, so they can be cached to avoid recomputation. The choice of when to use forward or reverse mode can be inferred by their computational complexity. When the number of inputs is low compared to outputs, the forward mode is preferable. Similarly, the reverse mode is favored if the number of outputs is greater.

Algorithm 2.2: ICP

```
Inputs: S = source points, T = target, n = max_iterations, eps =
    error tolerance
Output: Transformation Matrix
begin
    // Create kd-tree if T is point cloud.
    // Otherwise, BVH in case of mesh.
    // Should be cached if target does not change
    T_fast = accelerate_distance_queries(T)

    // Identity matrix
    transform = I

    for itt in range(max_iteration):
        // Use automatic differentiation (section 2.5) if possible
        cost, jacobian = compute_cost_auto_diff(S, T_fast)
        if cost < eps:
            return transform
        fi

        // Otherwise, update the source point cloud
        transform = minimize_cost(cost, jacobian)
        S = transform * S
    end
    return transform
end
```

2.6 Iterative Closest Point

Iterative Closest Point (ICP) [Y. CHEN and MEDIONI, 1992] is a pretty popular algorithm used for aligning a source point cloud to a similar target shape (point set registration). The target itself may be a point cloud or a continuous mesh. ICP works by iteratively aligning the source to the target by minimizing the cost function. Usually, the cost function is based on the sum of squared distance between the source ($l-2$ norm) and the target. The algorithm comprises the following steps:

- Pre-process the target shape for efficient distance queries by using a k-D tree (section 2.1.1) and using Bounding Volume Hierarchy (BVH) (section 2.8). The former is enough if the target is just a point cloud, but both are required if the target is a mesh.
- For each point in the source point cloud, find the closest association in the target. This step is usually done by finding a point on the target closest to the source point.
- Using the source and associated points set, find an isometric transformation matrix that minimizes the cost function.
- Apply the isometric transformation to the source point cloud.
- Repeat from the association step iteratively until convergence (which might not be the global minima).

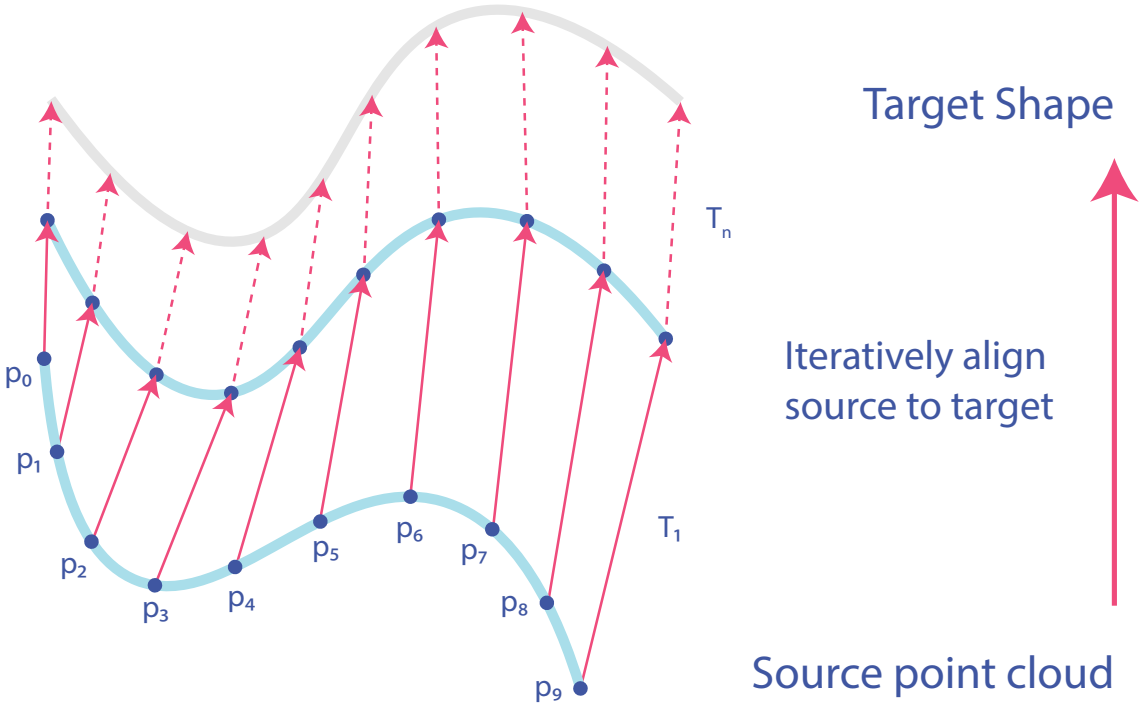


Figure 2.10: ICP aligning a point cloud to a continuous surface

The pseudo-code is presented in Algorithm 2.2. As ICP is a local refinement method, it is highly dependent on the quality of the initial guess. Even when the source and the target match exactly, there is no guarantee that the optimization will reach the global minima if the starting estimate is too far away. Another serious problem is the ability to deal with outliers. ICP generally performs adversely in the case of relatively many outliers as $l-2$ norm-based error increases quadratically with distance. This effect of outliers makes the optimization get stuck in local minima.

2.7 Robust loss functions

To guide the optimization properly and make it comparatively robust to outliers, better loss functions are needed instead of the $l2$ norm. There exist many such functions of which Huber loss [HUBER, 1992] is the most common one. It is a mixture of $l1$ and $l2$ loss used for robust regression. It can mathematically be represented using:

$$L_\delta\left(y, f\left(x\right)\right) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for } |y - f(x)| \leq \delta \\ \delta\left(|y - f(x)| - \frac{1}{2}\delta\right), & \text{otherwise} \end{cases} \tag{2.17}$$

Where δ is the scale parameter. It affects the transition point of Huber loss from $l2$ to $l1$ loss. One of the caveats on using the formulation in eq. (2.17) is that it can only be differentiated once. If a smoother approximation is required, the following form can be used [CHARBONNIER et al., 1997]:

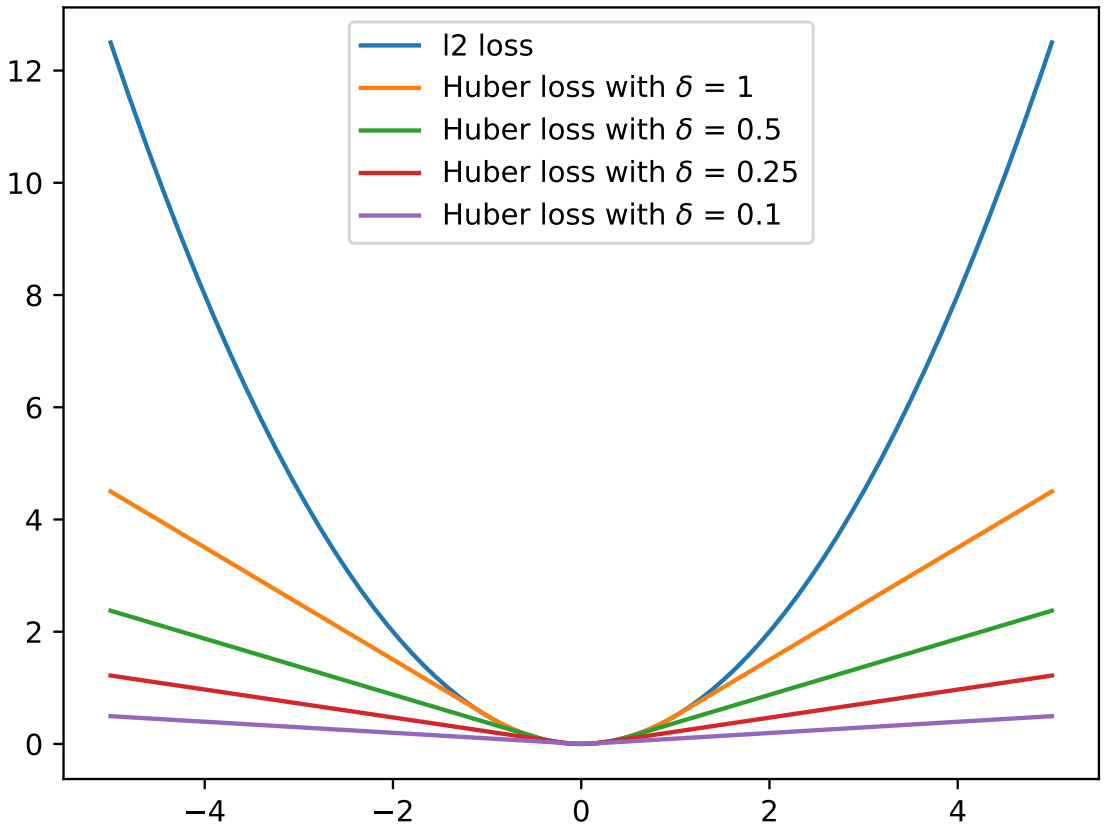


Figure 2.11: Huber loss at different values of δ compared to $l2$ loss

$$L_\delta\left(y, f\left(x\right)\right) = \delta^2\left(\sqrt{1+\left(\frac{\left(y-f(x)\right)}{\delta}\right)^2}-1\right) \tag{2.18}$$

It can be differentiated infinitely (C^∞ function) and might be used in schemes that require higher-order differentiability.

2.8 Bounding Volume Hierarchy

Bounding Volume Hierarchy (BVH) are a famous acceleration structure for speeding up intersection tests that are useful for efficient distance queries from a point to a mesh. A naïve way of checking where a ray intersects the given set of primitives is to sequentially go through all of them and select the closest one. Unsurprisingly, the computational complexity of this sequential approach is $O(n)$ where n is the number of primitives. This approach might be a reasonable option if the intersection test is required sparsely, but in most cases, where the test is performed often, it is better to create the proper data structure.

BVH wraps all primitives into a bounding volume (usually Axis Alligned Bounding Box (AABB)) and recursively applies the same procedure to the leaf node primitives until the termination criterion is met. Figure 2.12 shows a simple case where different primitives are

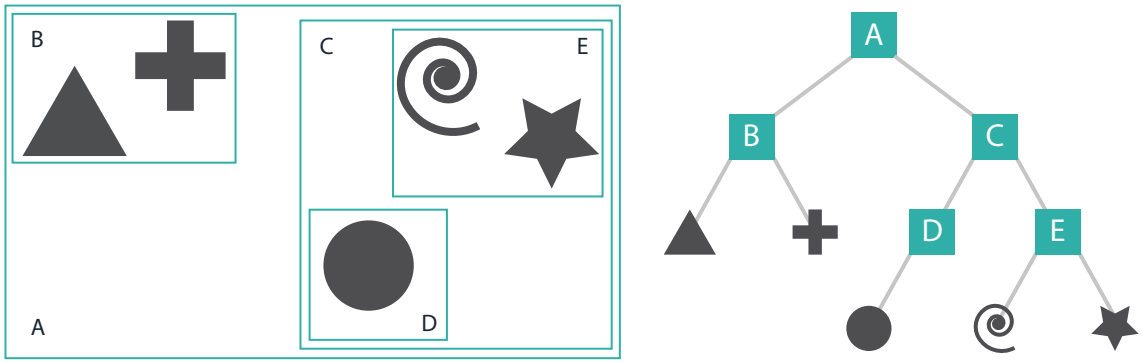


Figure 2.12: Left:BVH of an arbitrary set of primitives; Right: Corresponding transversal tree

wrapped using the mentioned methodology. This strategy reduces the search complexity down to $O \log n$. Finding the optimal BVH is an NP-hard problem [HAVRAN, 2000] and therefore tackled with heuristics. One such widespread technique is surface area heuristics [MACDONALD and BOOTH, 1990], which tries to minimize the surface area for most primitives, thus reducing the probability of an arbitrary ray hitting a specific AABB.

For distance computations, the process is as follows:

- Given a query point p_q, use the underlying k-D tree to find the closest point p_c.
- Create a bounding box from center point p_q to p_c.
- Use BVH to find intersecting bounding boxes recursively until the primitives are found.
- Find the closest primitive using the intersection test.

The underlying BVH should also contain the k-D tree to efficiently look up the nearest neighbor for best performance.

2.9 Neural Networks

As the name suggests, neural networks are a collection of artificial neurons. They are modeled after how human brains work in theory. According to the universal approximation theorem [HORNIK et al., 1989], they can approximate any function given enough neurons (and layers) though they might not generalize. This property allows them to represent arbitrarily complex functions provided enough data is available. With the increase in computational power, neural networks have become the de facto standard for most complicated problems that cannot be programmed explicitly. Such issues include image recognition, data processing, classification problems, and natural language processing, to name a few. The general architecture of a simple neural network is shown in fig. 2.13.

The problem at hand determines the number of inputs and outputs. The internal hidden layers are a design parameter. Increasing the number of inner layers makes it possible to

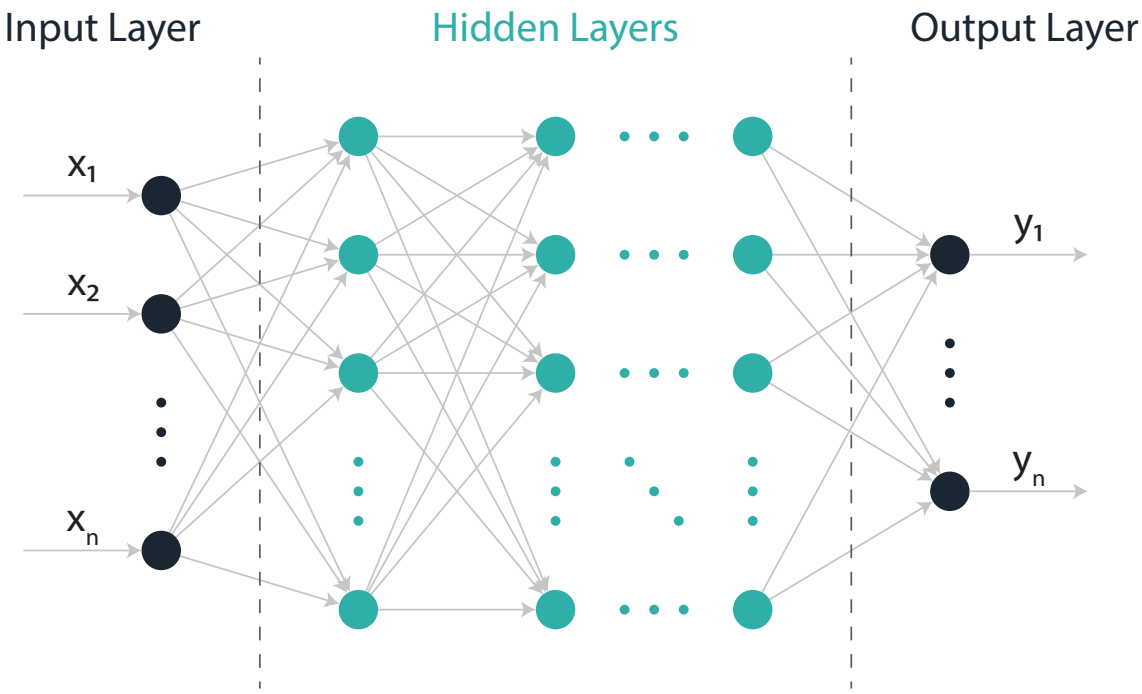


Figure 2.13: Architecture of a general neural network: The circles represent the neurons while the arrows depict connectivity and weights

learn more complicated problems. Too few hidden layers cause the network not to learn appropriately and have high errors. On the flip side, too many internal layers cause the network to over-fit the data, thus losing the ability to generalize to unseen situations. There are many heuristics to determine the near-optimum number of layers [KARSOLIYA, 2012] though there exists no method to compute such a number in a general setting.

To train the network, data is collected, associating inputs to outputs. The weights are initialized to random, and the result is predicted using the inputs. The predicted results are then compared to the actual outputs, and the loss is propagated back to update the internal weights. This backpropagation relies heavily on automatic differentiation discussed earlier in section 2.5. Machine learning and neural networks are vast fields. There are plenty of resources [GOODFELLOW et al., 2016 GÉRON, 2019 BURKOV, 2019...] that discuss these concepts in more detail.

Chapter 3

Related Work

This chapter gives a general overview of recent techniques for utilizing the BIM / CAD model into the SLAM problem. These approaches are grouped depending upon how they use the underlying environment model.

3.1 Occupancy Grid Methods

Accurate localization is a core requirement for autonomous driving. This has propelled the research immensely towards this direction in the robotics community during the past few decades. There exist plenty of methods to solve this problem that are relatively accurate and robust. Initial efforts developed methods that were pretty simple such as EKF [JULIER and UHLMANN, 1997] for non-linear systems and histogram filters [BURGARD et al., 1996]. These methods suffer from high errors. To solve this issue, probabilistic approaches based on particle filters such as Monte Carlo localization [FOX et al., 2001] were introduced. These approaches commonly model static parts of the environment in the form of an occupancy grid [ELFES, 1987]. In such a representation, rather than storing the obstacles in a continuous form, they are discretized and placed in discrete cells. The resolution of the cells is directly related to that of underlying sensors. As shown in fig. 3.1, a continuous environment is converted into any occupancy grid. This discretization makes the problem comparatively simpler for estimating states approximately [THRUN, 2002].

This concept can easily be extended into 3D [ELFES, 1989] using a voxel grid. Due to their simplicity and widespread use in the robotics community, such as Robot Operating

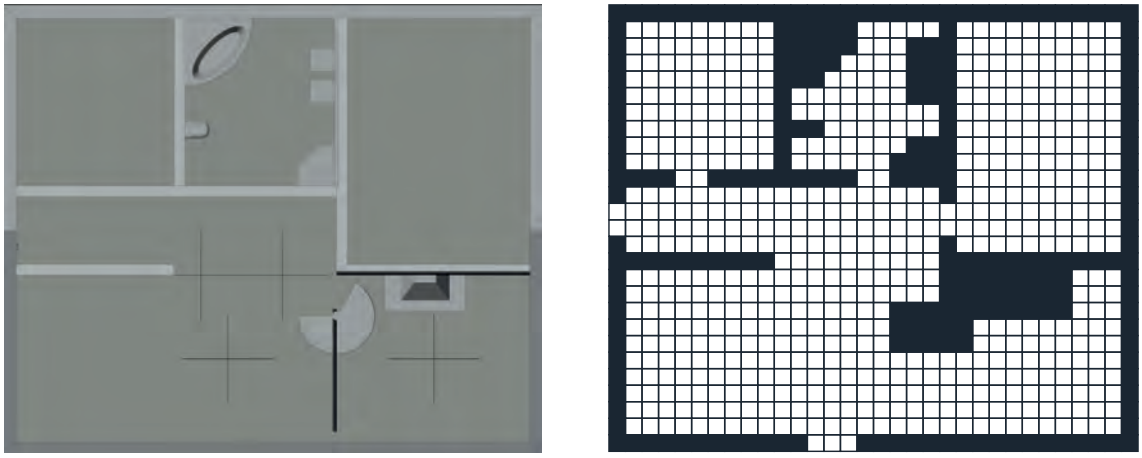


Figure 3.1: Left: Top view of an arbitrary house; Right: Equivalent 2D occupancy grid representation at ground level. Note that the obstacles above agent height are not included

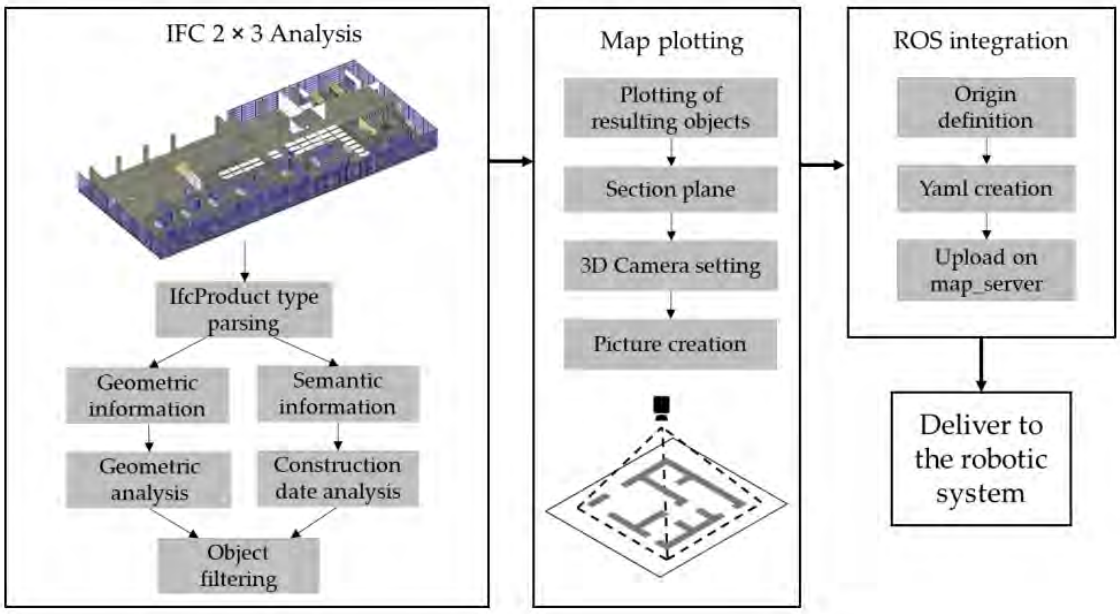


Figure 3.2: Time dependent maps proposed for integration with ROS [FOLLINI et al., 2021]

System (ROS) [STANFORD ARTIFICIAL INTELLIGENCE LABORATORY ET AL., 2020], a lot of approaches utilize the environment model in this form. For example, PRIETO et al., 2020 in order to evaluate the construction progress, use BIM model to extract the 2D floor plan directly if the underlying model contains the curve representation. In cases where such information is not available, the Boundary Representation (B-Rep) of the rooms is used to approximate the 2D map. This 2D floor plan is then used to generate the 2D occupancy grid. This information is then further used for navigation and later for comparison with the collected data [1]. Similar approaches have been conducted in the past that utilize the 2D map for evaluating construction progress and accuracy [BHATLA et al., 2012, PĂTRĂUCEAN et al., 2015, ARMENI et al., 2016].

A similar method is proposed by FOLLINI et al., 2021 which is summarized in fig. 3.2. One of the main differences between this and the aforementioned method is that this method aims at integration with ROS stack while generating time-dependent maps.

Naturally, there are 3D extensions as well. Recently, MOURA et al., 2021 proposed a procedure for integrating BIM model into google cartographer [HESS et al., 2016]. Apart from being used in localization, analogous methods have also been explored in navigation and path planning [LIN et al., 2013, XU et al., 2017, KARIMI et al., 2021, X. LIU et al., 2021].

3.2 Feature Based Methods

As the name suggests, these methods do not simply discretize the environment but rather use mesh features to improve the SLAM accuracy. One of the most popular ways

[1] The data is collected from the center point of each bounding volume

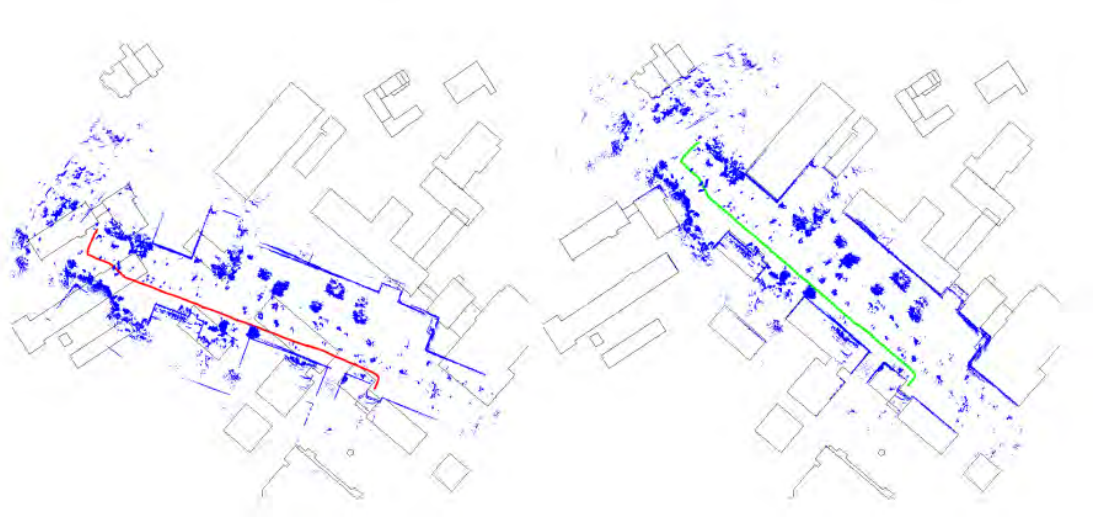


Figure 3.3: Using publicly available maps to align point cloud to buildings [VYSOTSKA and STACHNISS, 2016]

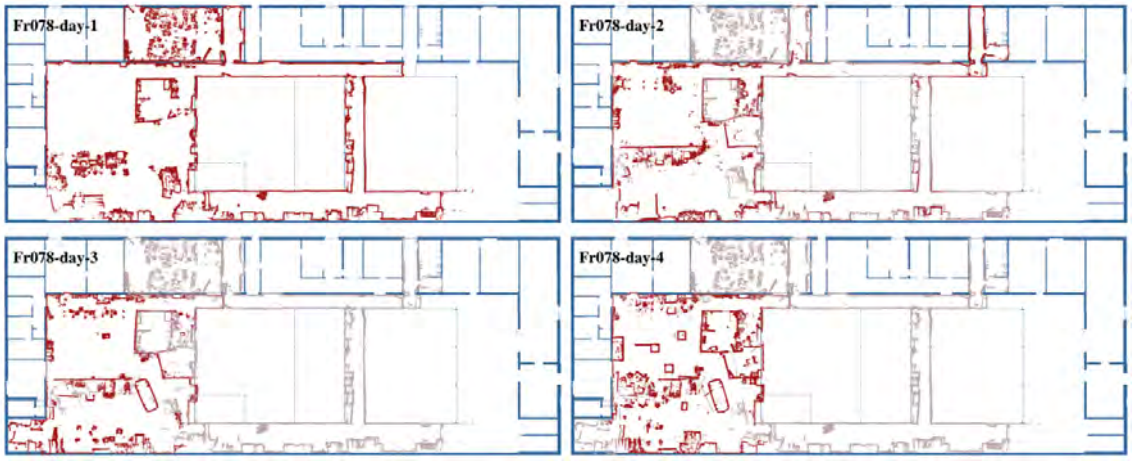


Figure 3.4: Dynamically updating map over the course of four days [BONIARDI et al., 2019]

is to match the point cloud to these mesh features using one of the variants of ICP. VYSOTSKA and STACHNISS, 2016 proposed a pose graph-based SLAM algorithm for outdoor navigation that indirectly incorporated building information. Leveraging publicly available maps and extracting building information and road networks (as shown in fig. 3.3) leads to notable error reduction.

Another approach that integrates CAD floor plans into graph-based SLAM is BONIARDI et al., 2019. For matching points and mesh features, generalized ICP is used along with robust kernels to reduce errors. One of the main features presented is that of long-term navigation. In most realistic scenarios, the environment is dynamic and may change significantly over time. A static map built with all past observations will be outdated and potentially lead to wrong correspondences. To fix this issue, the approach proposed by [BONIARDI et al., 2019] updates the built map. To this end, the built map is matched to the reading from LiDAR scan, and a probabilistic model is used to prune the outdated data in case of discrepancies. Figure 3.4 shows the methodology in action.

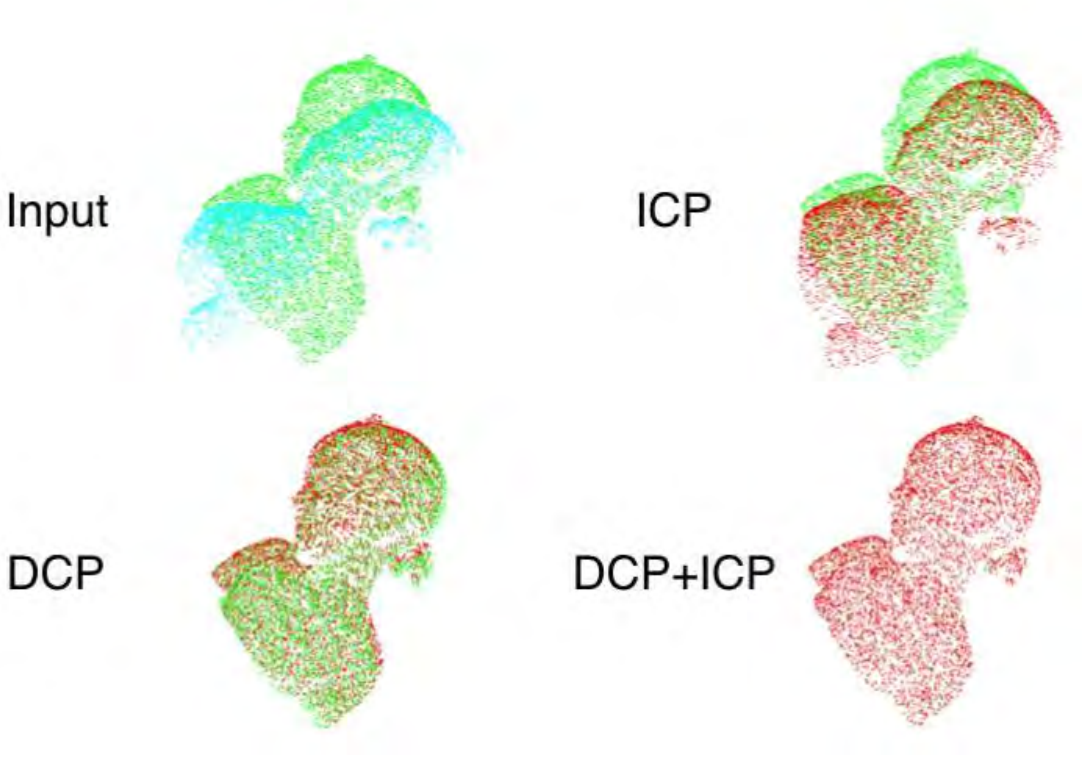


Figure 3.5: Matching source to target point cloud using Deep Closest Point (DCP) [WANG and SOLOMON, 2019]

A recent study [OELSCH et al., 2021] builds on top of the state-of-the-art LOAM algorithm. Rather than using the BIM model of the environment, it leverages a reference object with a known position in the global frame. By utilizing corner and surface features from LOAM and mesh features from points to mesh correspondences, it poses SLAM as a joint optimization problem. This problem is then solved to boost the localization precision. One distinct feature is that it works in a 3D setting, making it viable for aerial robots.

3.3 Deep Learning Methods

With the increase in computational power and advances in machine learning, deep learning techniques have become increasingly popular. There is quite a lot of variety depending upon how they utilize the model. They can generally be grouped into three broad categories: 2D to 2D, 2D to 3D, and 3D to 3D. The former two groups are mainly for cameras, whereas the last group deals with LiDAR sensors. Primarily, 3D to 3D methods for global registration will be discussed. There are, of course, hybrid approaches as well. For example, H. BLUM et al., 2020 uses a Convolutional Neural Network (CNN) to achieve foreground and background segmentation of the camera images of the perceived environment, which is then propagated to perform selective localization using ICP on the point cloud obtained from LiDAR.

[YIN et al., 2018] introduced LocNet to search scans in a prior global map by introducing a rotational invariance technique and learning the environment representation. One limitation

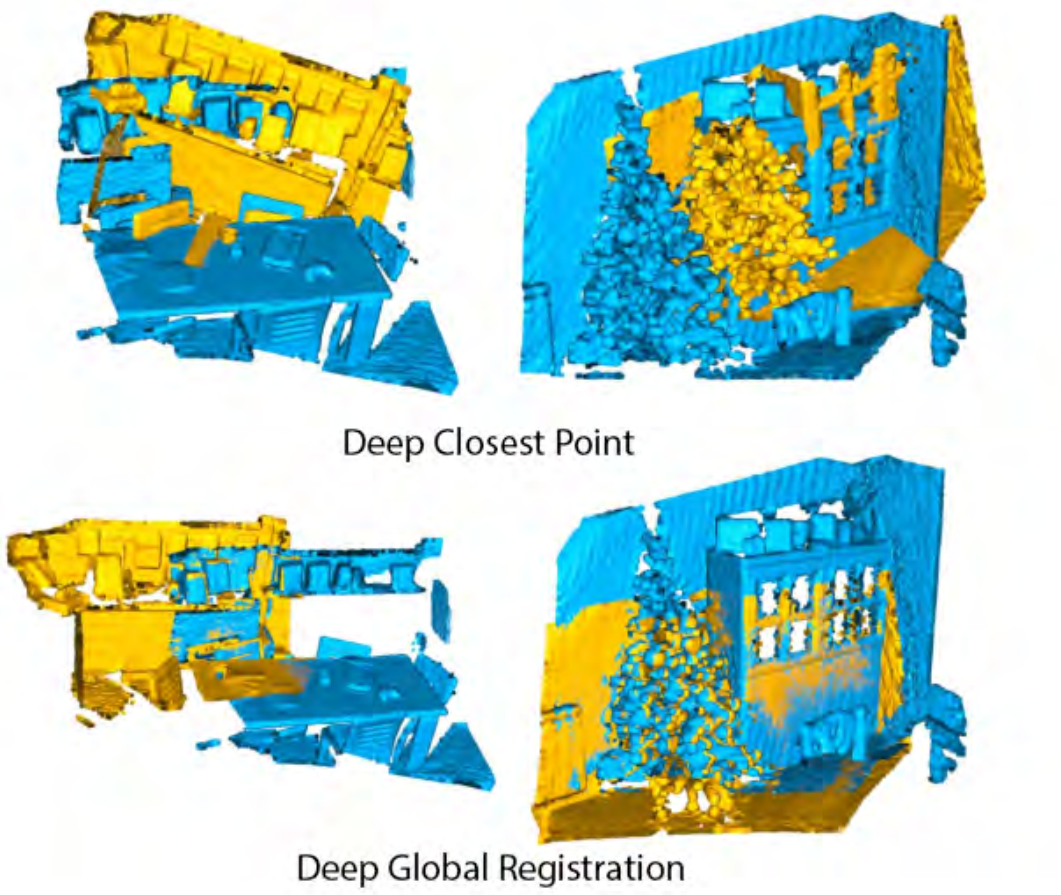


Figure 3.6: Comparison of DGR with DCP [CHOY et al., 2020]

of this approach is that it cannot generalize to unknown environments. Pointnetvlad [UY and LEE, 2018] overcame this limitation by learning global descriptors from the point cloud. Global registration usually requires a global descriptor to have encoded local features. PCAN [W. ZHANG and XIAO, 2019] improves on Pointnet by introducing an attention network to evaluate the relative significance of each local feature. This score is then used to give more importance to relevant features, thereby significantly improving the accuracy.

A deep learning alternative to the standard ICP is DCP [WANG and SOLOMON, 2019]. It improved on the accuracy by using a three-part network: point cloud embedding network, transformer-based layer, and a SVD layer. By using DCP for the initial alignment and standard ICP for fine-tuning (fig. 3.5), the solution converged to the global optima in really challenging situations.[2]

DGR CHOY et al., 2020 further improved the accuracy of DCP by utilizing the geometric features obtained by Fully Convolutional Geometric Features (FCGF) [CHOY et al., 2019] proposed by the same authors. Figure 3.6 shows the superior performance of this approach compared to its predecessor. This approach trains on two popular public datasets: Karlsruhe Institute of Technology and Toyota Technological Institute (KITTI) dataset and 3DMatch dataset. The voxel size selected for training FCGF is 30 cm and 5 cm

[2]A brief overview of deep learning techniques in SLAM is given by [C. CHEN et al., 2020] in more detail

respectively. This choice of size means that the final localization of DGR is dependent on this parameter as well.

3.4 Conclusions From Literature Review

After going through multiple approaches, an appropriate direction must be selected. Taking a look at the famous KITTI benchmark for odometry, the best performing LiDAR based algorithms are Visual Lidar Odometry And Mapping (V-LOAM) [J. ZHANG and SINGH, 2015] and LOAM [J. ZHANG and SINGH, 2014] respectively. The former incorporated visual odometry into the latter framework to improve accuracy. As V-LOAM is a superset of LOAM, any methodology developed on the base algorithm LOAM should be extendable to the other. Therefore, LOAM is selected as the base algorithm for SLAM computations. As already discussed in this chapter, there are multiple ways to integrate the BIM / CAD into the SLAM calculations. A methodology that is fast, robust, and compatible with LOAM is required.

Global localization algorithms are more expensive than local refinement methods. As agent navigation is a continuous process, i.e., the agent cannot physically teleport from one point to another in real life, robust local refinement methods should be able to handle the registration process given that the two consecutive scans happen relatively close in space. Therefore, solving a global registration problem is not necessary at every iteration. Nonetheless, there is still a need for such an algorithm when the agent starts in an unknown environment to compute the approximate initial pose. Chapter 4 discusses in detail more of the design decisions and how to extend LOAM algorithm to incorporate the environment model to improve accuracy robustly.

Chapter 4

Methodology

To robustly integrate the BIM model into the LOAM framework, the main contribution is added to the odometry part as increased localization accuracy leads to better mapping quality as well. An overview of the proposed methodology is shown in fig. 4.1. Section 4.1 goes over how the LiDAR scan is acquired, how the environment model requirements, and the overall underlying assumptions. The main odometry module is discussed in detail in section 4.2. As the suggested approach requires a reasonable initial estimate of the global pose of the agent to match mesh features, a robust deep learning methodology is employed, which is briefly illustrated in section 4.3. The mapping part of LOAM benefits directly from better odometry estimates though the internal working remains unchanged as explained in section 4.4. Collectively, the approach proposed in this thesis is referred to as Model Aware Lidar Odometry And Mapping (MA-LOAM).

4.1 Data Acquisition

To use the proposed approach, there are two main requirements, namely:

- An environment model should be present to extract the mesh features. In case it is missing, the methodology is equivalent to the standard LOAM implementation and therefore cannot benefit it.
- A series of LiDAR scans are available that have at least some overlap between adjacent scans so that they can be associated with each other. If there is no overlap, the odometry estimate might still converge if the scans are not too far away and have a relatively high degree of association to the environment model.

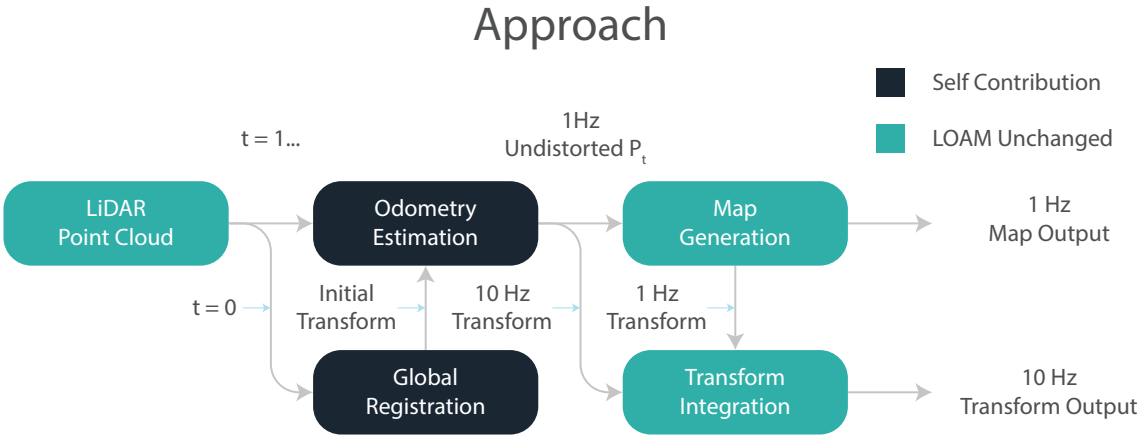


Figure 4.1: An overview of the proposed methodology

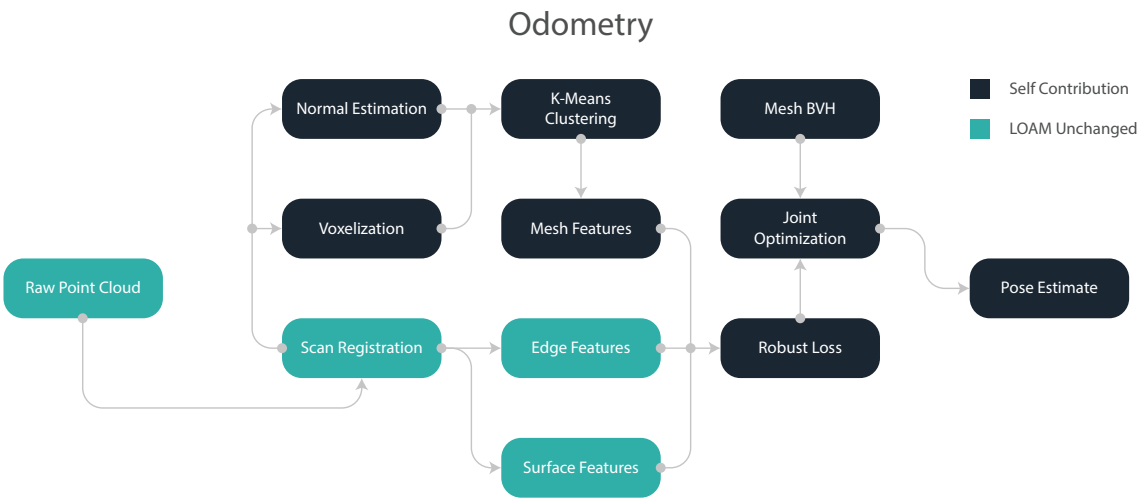


Figure 4.2: Pipeline for integrating BIM model into LOAM to improve odometry estimate

The LiDAR point cloud can be structured or unstructured depending upon the scan pattern. The proposed methodology works for both the cases though mainly the former is explored as LOAM itself extracts corner and surface features considering the point cloud is structured. As for the environment model, the algorithm is generally insensitive to scan model differences as outliers rejection is addressed at multiple levels throughout the odometry pipeline. This rejection allows for using models that are significantly different than the actual scan, given that the relatively more prominent features (like walls, ceiling, etc.) are somewhat observable.

4.2 Odometry

In this thesis, the most significant contribution is towards improving the odometry estimates. Figure 4.2 shows a general outline of the odometry pipeline. The algorithm extracts corner and surface features similar to LOAM. The mesh features are extracted robustly using a methodology similar to that of TAZIR et al., 2018. The main difference is that here, it is used as a preprocessing step to cluster points that are similar in a given vicinity. More details are presented in relevant sections.

4.2.1 Scan Registration

The point cloud coming from the LiDAR sensor is first preprocessed before being passed to the feature extraction pipeline. Points that lie too close or are too far away to the LiDAR sensor are removed as these points are considered to be of low fidelity as shown in fig. 4.3. As edge and surface features assume that the point cloud is structured, the point cloud is sorted into separate scan lines based on their azimuth angle ϕ relative to the LiDAR frame. Figure 4.4 depicts this process where the points are associated to ϕ_0, ϕ_1 or ϕ_2 depending upon their angle. Some LiDAR sensors have scan lines at equidistant angles, in which case, it is quite trivial to assign points to a scan line. If they are not equidistant and the

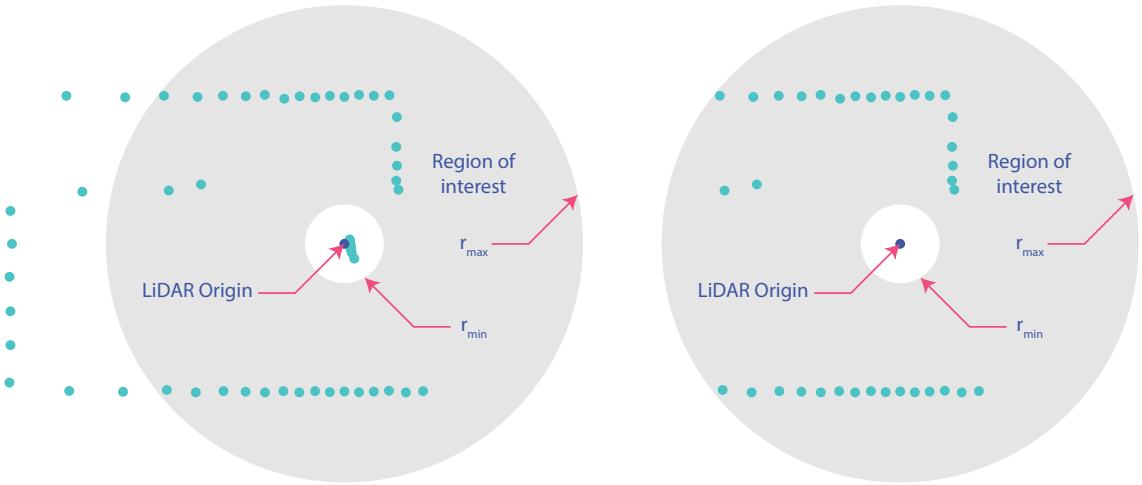


Figure 4.3: Points inside region of interest, that is, between r_{min} and r_{max} are retained. Left: Raw scan for LiDAR sensor; Right: Scan after filtering points outside region of interest

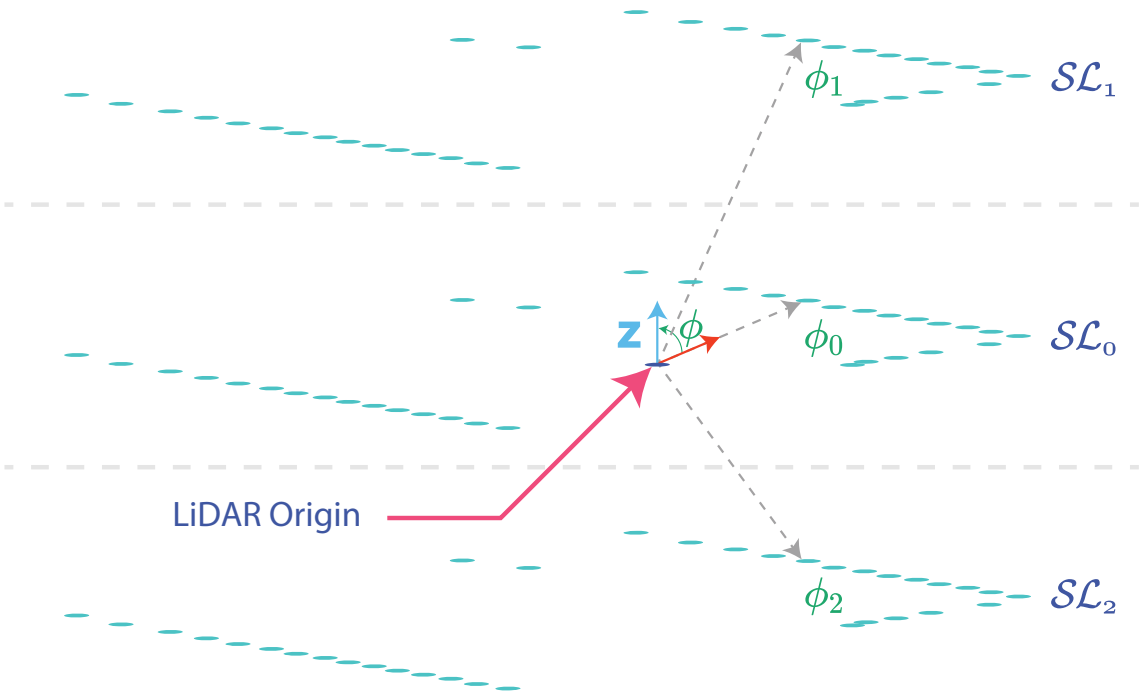


Figure 4.4: Points with almost the same azimuth angle ϕ are associated to separate scan lines $\mathcal{SL}_i$

sensor specification is available (scan lines angles are usually fixed and specified), the points can be associated with the nearest azimuth angle of the sensor. Lastly, a clustering algorithm can be used in a general case where only the number of scan lines are available.

4.2.2 Mesh Features Extraction

Extracting mesh features $\mathcal{F}_M$ is much more involved than edge and planer surface features (denoted using $\mathcal{F}_E$ and $\mathcal{F}_S$). Using all the points from the scan does not add much extra value compared to a more intelligent approach that uses only a subset of important points. Worse still, the dense point cloud can potentially contain many outliers and can thus lead to wrong associations. One standard methodology to downsample the point cloud is to use a voxel grid filter. That is, all points that lie within the same voxel are collapsed to a single point at their collective centroid (as shown in fig. 4.5, case A, simple). This naïve downsampling can lead to a loss in detail and, therefore, poor feature extraction. To avoid this, the proposed algorithm starts by taking the registered scan (from section 4.2.1) and putting them inside a 3D cartesian voxel grid as described in section 2.3. Then, the empty voxels are removed. Concurrently, the surface normals are estimated using the methodology discussed in section 2.2 using a specified number of nearest neighbors. Now, instead of collapsing all the points within the voxel, the points are clustered based on their normals using k-means clustering. Points that are part of the same cluster are collapsed together, thus preserving the extra detail (as depicted in fig. 4.5, case A, k-means). The normals of two different surfaces may point in the same direction. If the points are then collapsed using k-means clustering, the features can be lost for some cases (e.g., fig. 4.5, case B, k-means). If the voxel size is small, this approximation still works well in practice, but in case the details must be preserved, more advanced plane based clustering algorithms [SHAO et al., 2013, Z.-M. YANG et al., 2015, L.-M. LIU et al., 2017] can be used to preserve the detail. For example, the k-proximal plane clustering takes into account the proximity of the fitted plane along with the surface normals computed somewhat similar to the eigenvalue-based methodology already discussed in section 2.2. The result is shown in fig. 4.5, case B. It is also possible to fit higher-order shapes to account for curvature, for example, but the slight improvement in accuracy usually does not justify the extra computational effort. One of the main reasons is that at a small voxel size ($\sim 10cm$), a linear approximation (plane) is not too different from a higher-order estimate.

To remove the outliers originating from random noise, clusters with cardinality below a certain threshold can be dropped. So far, to collapse the points, clustering algorithms are used. Usually, the optimal number of clusters is unknown and must be determined algorithmically. To this end, silhouette coefficient [ROUSSEEUW, 1987] is used given by:

$$s(i) = \begin{cases} 1 - a(i)/b(i), & \text{if } a(i) < b(i) \\ 0, & \text{if } a(i) = b(i) \\ b(i)/a(i) - 1, & \text{if } a(i) > b(i) \end{cases} \tag{4.1}$$

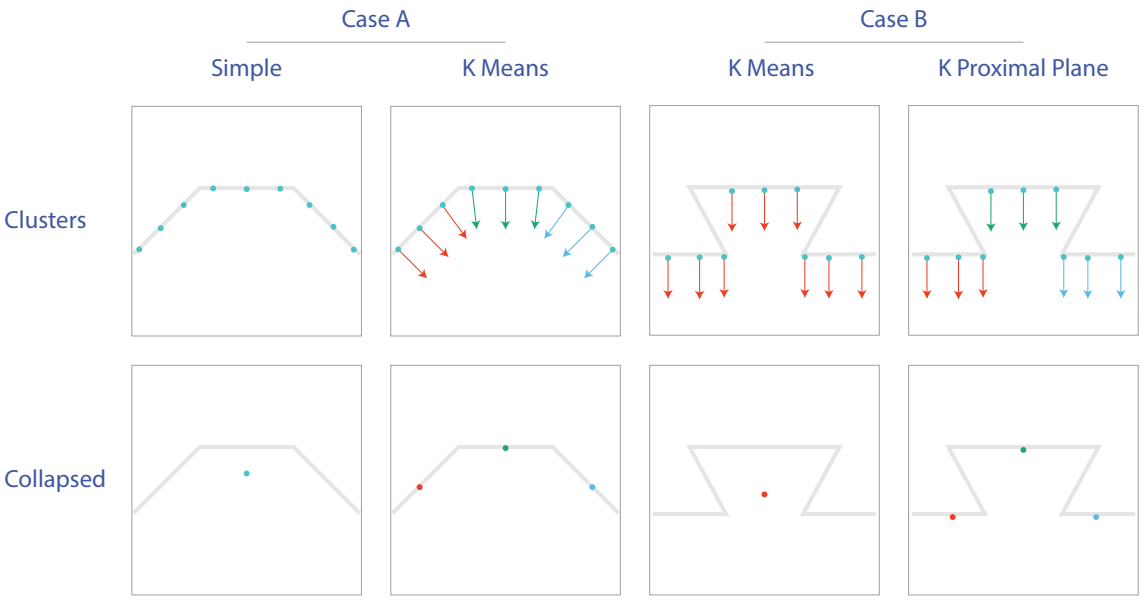


Figure 4.5: Different clustering techniques employed to retain features; Case A: A simple collapse results in loss of detail. Collapsing based on k-means clustering applied to surface normals works well in most cases; Case B: A case where k-means based on surface normals might lead to a loss in detail. A more advanced technique like k-proximal planes might be used to retain the features

Or in a more concrete fashion:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{4.2}$$

where a represents the mean distance of any point i to its own cluster where as b represents the smallest mean distance to cluster that i does not belong to. Mathematically, they are given by:

$$a(i) = \frac{1}{|C_I| - 1} \sum_{j \in C_I, i \neq j} d(i, j) \tag{4.3}$$

$$b(i) = \min_{J \neq I} \frac{1}{|C_J|} \sum_{j \in C_J} d(i, j) \tag{4.4}$$

In the case of a single cluster, the metric is set arbitrarily to zero. As this clustering metric can be computed for two clusters and more, the case of a single cluster as an optimum is handled explicitly. If the silhouette score indicates that two clusters are optimum, then those clusters are merged within a set epsilon. These clustered points computed using the said methodology form the basis of finding mesh features correspondences. Figure 4.6 shows the process of collapsing points on an actual LiDAR scan. Additionally, there can be scenarios where the number of mesh features is too high. In that case, mesh features originating from small clusters can be dropped while taking into account the relative distance of the cluster from the LiDAR sensor.

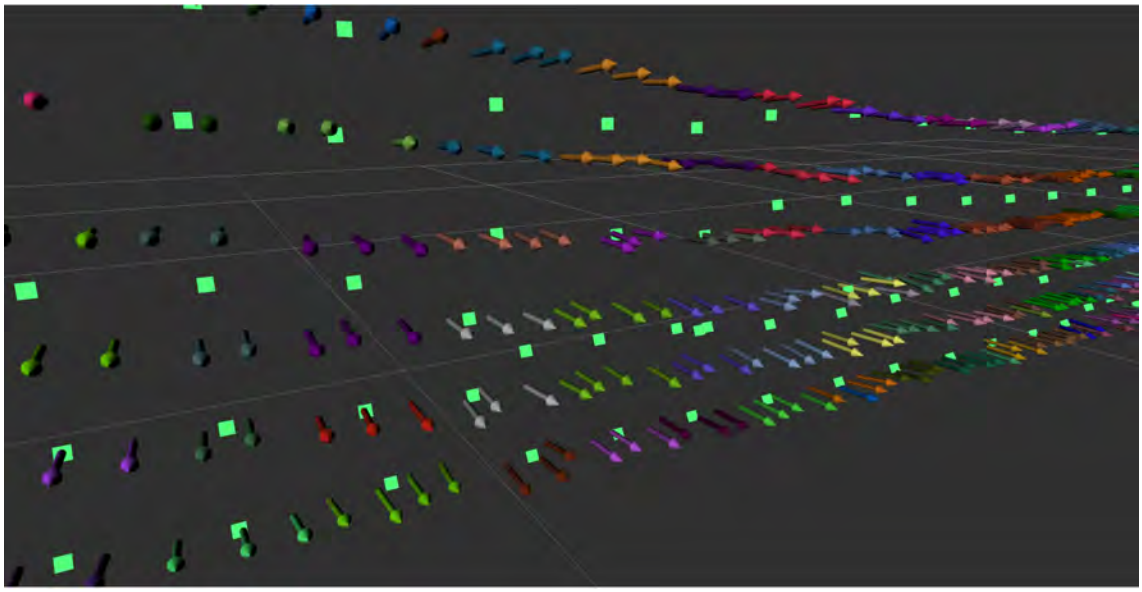


Figure 4.6: Estimated normals of a planer surface: Points belonging to the same voxel have same colored normals. The collapsed point cloud is shown using flat square

4.2.3 Edge And Surface Features Extraction

The procedure for extracting the edge and surface features ($\mathcal{F}_E$ and $\mathcal{F}_S$) is the same as the original method proposed in LOAM. Let i be a point belonging to a specific scan line. As the LiDAR scanner rotates in a particular direction, that is, either clockwise or counterclockwise, $\mathcal{S}$ will contain half of the points on each side of the point i. The local smoothness of the point i can be computed using[1]:

$$c = \frac{1}{|\mathcal{S}| \cdot \left\| \boldsymbol{X}^L_{(k,i)} \right\|} \left\| \sum_{j \in \mathcal{S}, j \neq i} \left(\boldsymbol{X}^L_{(k,i)} - \boldsymbol{X}^L_{(k,j)} \right) \right\| \tag{4.5}$$

The same process is repeated for the rest of the points in the scan. If the same number of neighbors are used for all the points, the cardinality term $|\mathcal{S}|$ may be dropped from eq. (4.5). The points are then sorted based on their c value. A high c value indicates an abrupt change in spatial coordinates, thus representing corner features. On the contrary, a low value of c indicates plane surfaces, thus indicating surface features. The scan is divided into multiple sub-regions to avoid features cluttered in space, usually 4 or 6. This division allows the features to be more uniformly distributed. Furthermore, each sub-region can only provide two edge and four planer surface features. Hence, a feature is dropped if its c value is outside a threshold range (as it indicates a poor feature) or the maximum number of feature points is already selected. Special care is also taken to avoid planer surface features that lie on a plane parallel to the LiDAR sensor laser. Similarly, edge features that might occur due to occlusion are also dropped. This process is shown in fig. 4.7. Finally, despite the c score, surrounding points to an already selected point are not considered

[1] Similar to LOAM, the notation $X^L_{(k,i)}$ means an arbitrary point i in scan k in the LiDAR frame L. World frame is represented using W

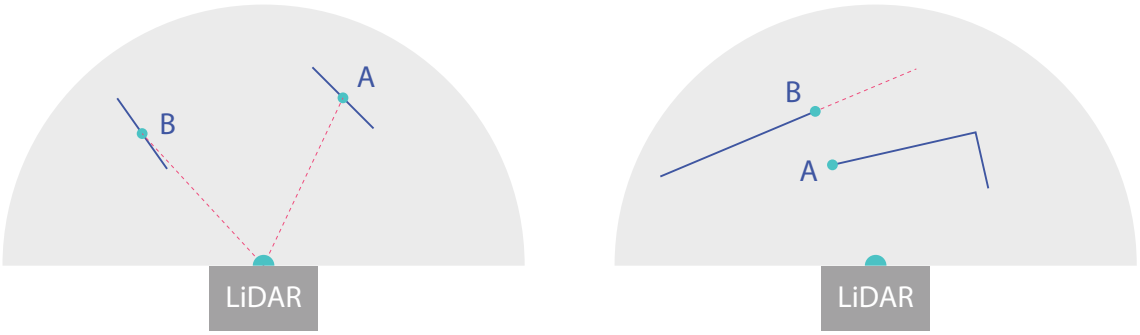


Figure 4.7: In both the cases, feature A is reliable where as feature B is considered unreliable; Left: Surface feature B is considered bad because it is parallel to LiDAR laser; Right: Corner feature B is unreliable as it is occluded. Moving the LiDAR sensor will change its position; Adapted from J. ZHANG and SINGH, 2014

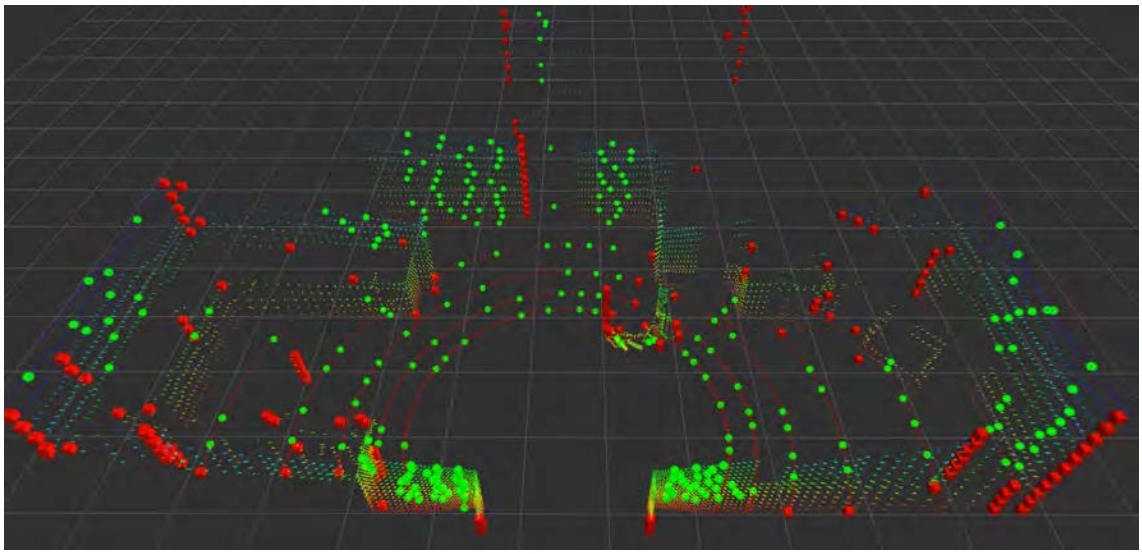
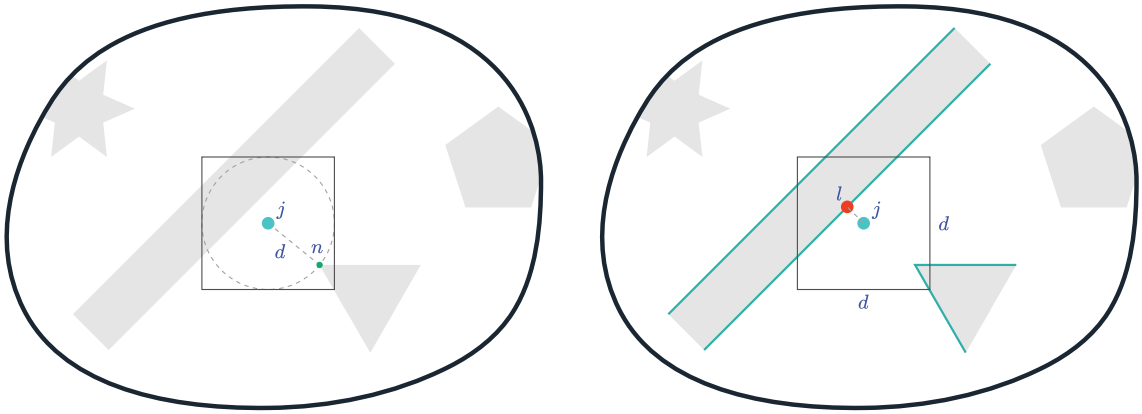


Figure 4.8: A sample indoor point cloud generated by a LiDAR scanner. The red cubes represent the edge features where as the green spheres represent plane surface features. Note that the surface features are twice as many as the corner features by choice

valid. It should be noted that these features are not necessarily temporally stable for data coming from an actual LiDAR scan even when the agent is not moving. The main reason for this is sensor noise, as minute differences can affect the c value of points, thus leading to different features. Nonetheless, this feature extraction is shown on an actual scan in fig. 4.8.

4.2.4 Joint Optimization

Finding good and reliable features constitutes the first half of the approach. The methodology discussed in this subsection deals with finding reliable correspondences in subsequent scans and formulating an optimization problem to solve for relative pose estimates. The process of finding correspondences is discussed below:



(a) Given a mesh feature j, find the nearest neighbor vertex n with distance d. Construct an AABB of length, width and height d, then compute intersection with the environment AABB tree (not shown to avoid cluter)

(b) Ideally, only the green highlighted edges must be queried for nearest point to mesh correspondence. The point l forms the correspondence pair (j, l) which is then used in the joint optimization

Figure 4.9: Finding a mesh correspondence point l for a mesh feature j using k-D tree for nearest neighbor search and AABB for intersection tests

- **Mesh correspondences:** With $\mathcal{F}_{\mathcal{M}}$ already computed, they must be matched to the available environment model. For that, the environment is stored inside a BVH namely an AABB tree. The corresponding k-D tree is also constructed to accelerate the neighbor's search. Collectively, both the acceleration structures along with the environment model are represented using $\mathcal{M}$. At each time step, the feature set $\mathcal{F}_{\mathcal{M}}$ is re-projected using the currently estimated transform. This projected feature set is denoted using $\tilde{\mathcal{F}_{\mathcal{M}}}$. Let i be a point in $\tilde{\mathcal{F}_{\mathcal{M}}}$. For each point, i, transform it from the local to the global frame using the past computed local transforms. This point is represented by j. For all points j, find the closest point l on $\mathcal{M}$. This feature pair (j, l) forms the correspondence of i. This process is shown in fig. 4.9. The cost function is then to find a local transformation (rotation R and translation T) that minimizes the following distance:

$$d_{\mathcal{M}} = \left(R^W \tilde{\boldsymbol{X}}^L_{(k,i)} + T^W\right) - \bar{\boldsymbol{X}}^W_{(k,l)} \tag{4.6}$$

  Where R^W and T^W are the currently estimated transformation to convert a point from the LiDAR frame to the world frame.

- **Edge correspondences:** Similarly, with edge features $\mathcal{F}_{\mathcal{E}}$ computed, correspondence must be selected. Let i be a point in the re-projected edge feature set $\tilde{\mathcal{F}_{\mathcal{E}}}$. Let j be the closest point (using k-D tree) of i in the current re-projected point cloud $\bar{\mathcal{P}}$. Let l be the nearest neighbor in two consecutive scans of j. This feature pair (j, l) forms the correspondence of i. It is explicitly made sure that both j and l are indeed corner features using eq. (4.5). The cost function is then to find a local transformation that minimizes the following distance:

$$d_{\mathcal{E}} = \frac{\left|\left(\tilde{\boldsymbol{X}}^{L}_{(k+1,i)} - \bar{\boldsymbol{X}}^{L}_{(k,j)}\right) \times \left(\tilde{\boldsymbol{X}}^{L}_{(k+1,i)} - \bar{\boldsymbol{X}}^{L}_{(k,l)}\right)\right|}{\left|\bar{\boldsymbol{X}}^{L}_{(k,j)} - \bar{\boldsymbol{X}}^{L}_{(k,l)}\right|} \tag{4.7}$$

- **Surface correspondences:** Finally, to define a planer surface feature, a point i is selected from the re-projected surface feature set $\tilde{\mathcal{F}}_{\mathcal{S}}$. Similar to edge features, the closest point j is selected from $\bar{\mathcal{P}}$. Then, two more closest points l and m are selected. Point l is the closest neighbor of i in the same scan as j while m is the nearest neighbor in two consecutive scans of j. Similar to edge features, eq. (4.5) is used to ascertain the validity of the selected points. The cost function can then be formulated as:

$$d_{\mathcal{S}} = \frac{\left|\begin{array}{c}\left(\tilde{\boldsymbol{X}}^{L}_{(k+1,i)} - \bar{\boldsymbol{X}}^{L}_{(k,j)}\right) \\ \left(\left(\bar{\boldsymbol{X}}^{L}_{(k,j)} - \bar{\boldsymbol{X}}^{L}_{(k,l)}\right) \times \left(\bar{\boldsymbol{X}}^{L}_{(k,j)} - \bar{\boldsymbol{X}}^{L}_{(k,m)}\right)\right)\end{array}\right|}{\left|\left(\bar{\boldsymbol{X}}^{L}_{(k,j)} - \bar{\boldsymbol{X}}^{L}_{(k,l)}\right) \times \left(\bar{\boldsymbol{X}}^{L}_{(k,j)} - \bar{\boldsymbol{X}}^{L}_{(k,m)}\right)\right|} \tag{4.8}$$

With the ability to compute the correspondences for all mesh, edge, and surface features, it is now possible to pose a joint optimization problem to solve for the actual local transformation of the sensor in between the scans. Rather than using Euler angles, quaternions are the preferred choice as they are not prone to the problem of gimbal locking. Special care is taken to parameterize the quaternions such that there are only three degrees of freedom instead of four by ensuring that the magnitude of the resultant quaternion is always 1. This parameterization avoids adding an extra optimization parameter while simultaneously ensuring that the quaternion is always normalized. The joint optimization problem using eq. (4.6), eq. (4.7) and eq. (4.8) that correspond to mesh, edge and surface features then becomes:

$$\begin{aligned} f\left(Q,T\right) = \frac{\lambda_{\mathcal{M}}|\mathcal{F}_{\mathcal{M}}|}{\Lambda_{\mathcal{C}}} \sum_{i\in\mathcal{F}_{\mathcal{M}}} \rho\left(\left\|d_{\mathcal{M}}\left(Q,T\right)_i\right\|^2\right) + \frac{\lambda_{\mathcal{E}}|\mathcal{F}_{\mathcal{E}}|}{\Lambda_{\mathcal{C}}} \sum_{i\in\mathcal{F}_{\mathcal{E}}} \rho\left(\left\|d_{\mathcal{E}}\left(Q,T\right)_i\right\|^2\right) + \\ \frac{\lambda_{\mathcal{S}}|\mathcal{F}_{\mathcal{S}}|}{\Lambda_{\mathcal{C}}} \sum_{i\in\mathcal{F}_{\mathcal{S}}} \rho\left(\left\|d_{\mathcal{S}}\left(Q,T\right)_i\right\|^2\right) \end{aligned} \tag{4.9}$$

where ρ is the selected loss function (for e.g. Huber loss, section 2.7), $|\mathcal{F}|$ represents the cardinality of a feature set and λ represent lagrange multiplier allowing different weights for different features. Lastly, $\Lambda_{\mathcal{C}}$ is the normalizing factor computed using:

$$\Lambda_{\mathcal{C}} = \lambda_{\mathcal{M}}|\mathcal{F}_{\mathcal{M}}| + \lambda_{\mathcal{E}}|\mathcal{F}_{\mathcal{E}}| + \lambda_{\mathcal{S}}|\mathcal{F}_{\mathcal{S}}| \tag{4.10}$$

Alternatively, the terms $|\mathcal{F}|$ and $\Lambda_{\mathcal{C}}$ might be dropped altogether to directly control the feature contributions using just the lagrange multipliers. If the value of $d_{\mathcal{M}}$ for a certain mesh feature is above a certain threshold distance, it is dropped from the optimization problem. The eq. (4.9) is finally solved using the Levenberg-Marquardt algorithm described in section 2.4.3 to obtain the relative pose estimate.

4.3 Global Registration

The methodology discussed in section 4.2 is dependent upon mesh features to improve the odometry estimates. To match a mesh feature, the estimated position of the agent in the global frame is required. This estimate does not need to be exact, but the closer it is to the ground truth, the more reliably mesh features can be computed. For this global registration, Deep Global Registration (DGR) is used.

The approach is carefully selected after testing multiple methodologies. Most standard global registration techniques do not generalize well to an arbitrary environment. DGR trained on KITTI dataset performs reasonably well. As DGR is a global registration technique that is trained on a $30\,\mathrm{cm}$ voxel grid for the said dataset, the global pose estimate is near the actual poses but not precise. To refine this estimate, the initial global pose estimate is used as a prior for a local refinement method, namely the cluster ICP already discussed in the odometry module.

4.4 Mapping

The mapping module is unchanged from the LOAM implementation. At the end of each odometry iteration, a pose estimate is produced along with an undistorted point cloud. The mapping module takes these and stores them in world coordinates. While the odometry estimation is performed at $10\,\mathrm{Hz}$, the mapping process happens at a lower rate of $1\,\mathrm{Hz}$. The features are extracted similar to that described in the odometry estimate. The main difference is that here, the mesh features are not considered. The mapping module also considers 10 times more features to align the pose estimates. The point cloud is stored in a voxel grid of $10\,\mathrm{m}$ areas, and correspondences are found using the eigenvalue decomposition of the covariance matrix of the surrounding points. For an edge feature, the eigenvalue will be dominant in only one direction, whereas it will be dominant in two directions for a planer surface feature. The cost function is based on eq. (4.7) and eq. (4.8) and solved using Levenberg-Marquardt algorithm with robust loss functions. A voxel grid filter of size $5\,\mathrm{cm}$ is used to avoid storing all the points.

Chapter 5

Testing And Validation

This section performs a validation study to ensure that the proposed methodology works well in practice. The results are then compared to the standard LOAM approach as well as cluster ICP and evaluated for improvement. The test cases are also run when there are significant differences between the LiDAR scan and the actual environment model. The limitations of the proposed methodology are also illustrated. Finally, a computational time study is conducted by changing the voxel size to match the mesh features.

5.1 Experimental Setup

The experiments and validation require access to LiDAR scans as well as the actual ground truth location of the agent. For this end, the whole environment is simulated in Gazebo [KOENIG and HOWARD, 2004] allowing virtual LiDAR scans as well as the absolute poses of the agent. The LiDAR sensor used for all scans is Velodyne VLP-16. Furthermore, normally distributed noise is added to the scan values to reflect reality better. The implementation makes extensive use of functionality provided by ROS[1] and the PCL library[2]. For the LOAM algorithm, publicly available implementation[3] is used as a starting point. The proposed methodology is added on top of it. The optimization problem is formulated and solved using Ceres solver[4]. The full implementation of this thesis is also publicly available[5]. Evo-tools [GRUPP, 2017] are used to compare the odometry estimates. In cases where the transformation between the odometry and world frame is available, the trajectories are compared without any alignment. Otherwise, Umeyama alignment [UMEYAMA, 1991] is used to determine the translation and rotation without scale variation.

5.2 Hyperparameters

The proposed methodology has multiple hyperparameters that can affect performance significantly. They are not changed in all the experiments unless explicitly specified. Finetuning is possible and can potentially lead to better results, but it is avoided for an unbiased evaluation of the proposed methodology. Table 5.1 shows the values of these hyperparameters used for the experiments, along with a brief description. The hyperparameters of LOAM are unchanged and used directly without any modification.

[1] https://github.com/ros/ros
[2] https://github.com/PointCloudLibrary/pcl
[3] https://github.com/HKUST-Aerial-Robotics/A-LOAM
[4] https://github.com/ceres-solver/ceres-solver
[5] https://github.com/darkscyla/MA-LOAM

Table 5.1: List of hyperparameters and their values for showcased experiments

Parameter	Value	Description
$d_{\mathcal{M}\ th}$	2 m	Maximum threshold distance above which mesh feature is dropped
n_{nn}	10	Number of nearest neighbors to consider for normal computation
$n_{\mathcal{F}_\mathcal{M}}$	2	Maximum mesh features to retain per voxel
$n_{pt\ min}$	2	Minimum points required for a cluster to be valid
l_{vox}	0.25 m	Size of the voxel
ϵ_{merge}	0.25	Norm difference below which normals of two different clusters maybe merged
$\lambda_\mathcal{E}$	1.0	Weight for edge / corner features
$\lambda_\mathcal{S}$	1.0	Weight for planer surface features
$\lambda_\mathcal{M}$	0.5	Weight for mesh features
$n_{\mathcal{F}_\mathcal{M}\ max}$	∞	Maximum number of mesh features to retain globally. Low cardinailty clusters can be dropped in favor of more dominant features

5.3 Haus FZK

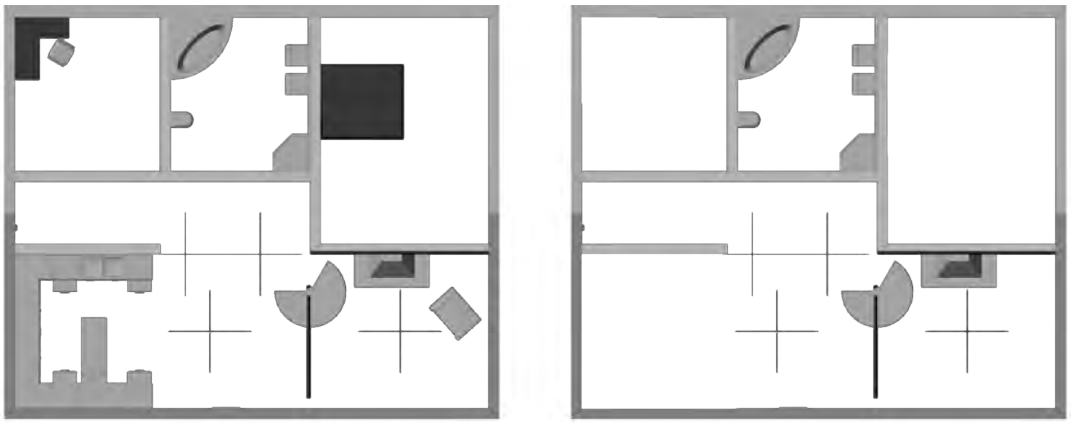
The environment model is shown in fig. 5.1a. This environment model is also used for LiDAR collisions. Similarly, fig. 5.1b illustrate the environment model without the furniture. This environment is simulated directly in Gazebo while a pre-computed trajectory (fig. 5.1c) is played to obtain the LiDAR scans. The cases can be divided into two categories depending on which model is used for mesh correspondences.

5.3.1 Case A: Same models

In this case, the same model (fig. 5.1a) is used for both the collision and mesh correspondences. This case is pretty straightforward and presented just for demonstration. A standard ICP algorithm should also be able to handle such cases as there are no scan model differences. The trajectory shown in fig. 5.1c is utilized to obtain the LiDAR scans which are then used to evaluate the performance of MA-LOAM, cluster ICP and LOAM.

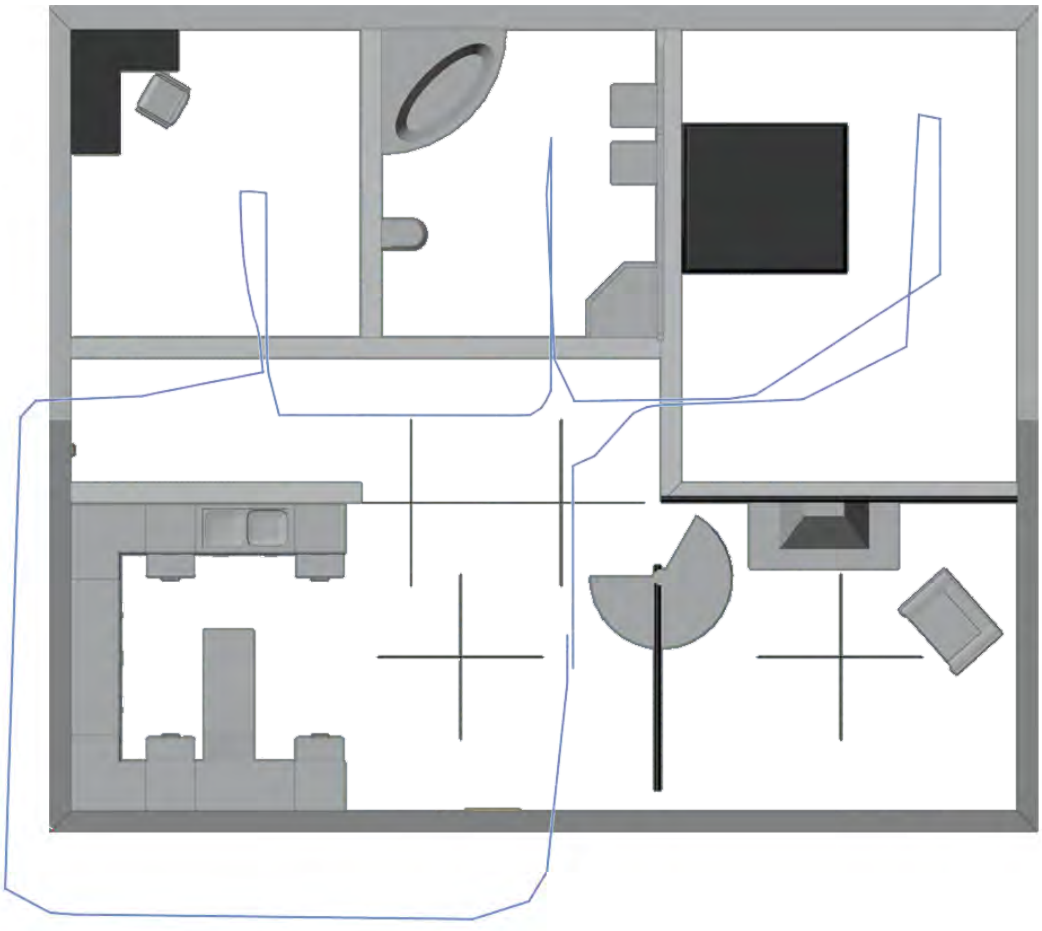
Figure 5.2 shows the estimated paths by respective algorithms. As there are no scan model differences along with the lack of dynamic elements, it is the ideal scenario for ICP algorithms. This is also verified by fig. 5.2.

Even though cluster ICP performs pretty well in this ideal case, it tends to exhibit oscillatory behavior in general. Also, it is worthwhile to note that the maximum error of cluster ICP is also relatively higher than MA-LOAM.



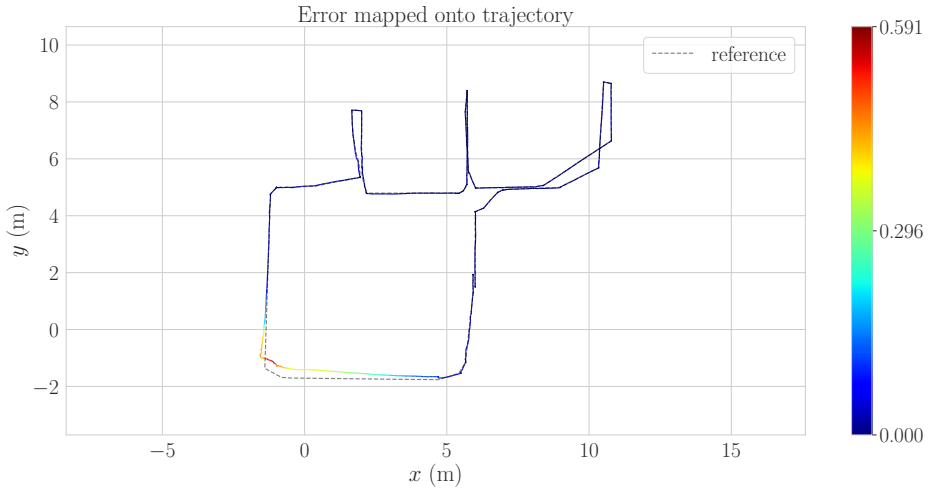
(a) Environment model of Haus FZK with furniture. This model is used for LiDAR collisions as well

(b) Environment model of Haus FZK without any furniture

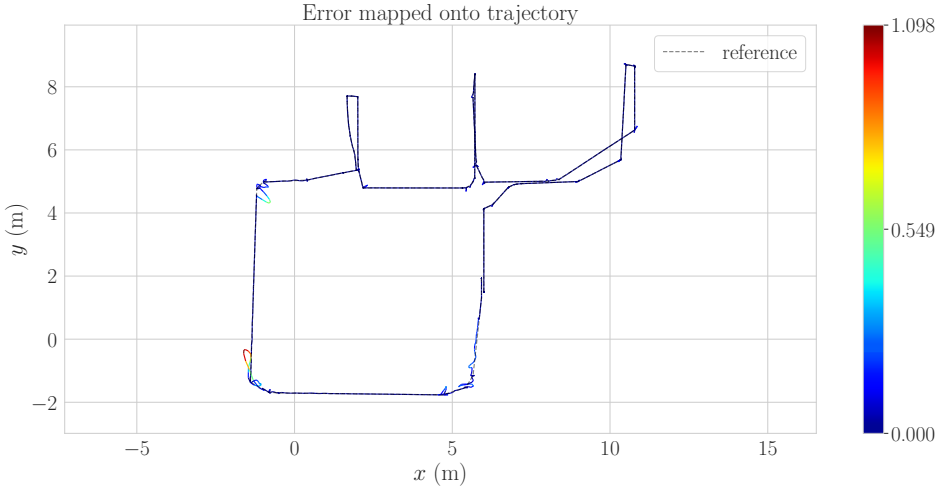


(c) Agent trajectory for environment Haus FZK. The agent moves around the house and returns to almost the same position. This trajectory is used as a baseline, and results are compared to the estimated path of the odometry algorithm. The trajectory length is $\sim$54 m. The average translational speed is around 1 m/s while the rotational speed is around 0.5 rad/s

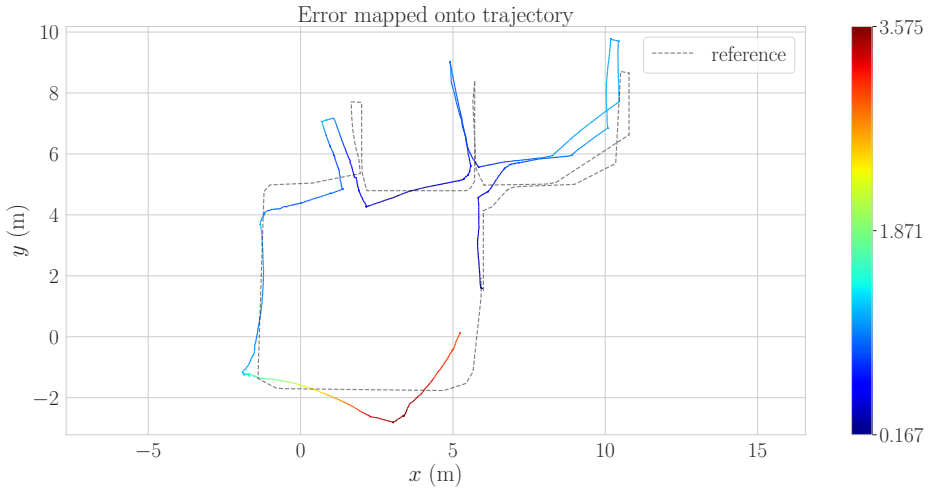
Figure 5.1: Haus FZK: Environment models along with the agent trajectory



(a) Path estimated by MA-LOAM



(b) Path estimated by cluster ICP



(c) Path estimated by standard LOAM

Figure 5.2: Haus FZK, Case A: Comparison of the agent path estimated by different algorithms when collision and mesh correspondences model is the same

Table 5.2: Haus FZK, Case A: Error statistics for different algorithms when the collision and mesh correspondences model is the same

	Error Metric	MA-LOAM	Cluster ICP	LOAM
$Translation\ (m)$	ϵ_{max}	$\mathbf{5.9068 \times 10^{-1}}$	1.0983	3.5747
	ϵ_{mean}	5.9768×10^{-2}	$\mathbf{2.8855 \times 10^{-2}}$	1.2260
	ϵ_{median}	1.5303×10^{-2}	$\mathbf{9.7140 \times 10^{-3}}$	1.0199
	ϵ_{min}	4.0700×10^{-4}	$\mathbf{1.8400 \times 10^{-4}}$	1.6656×10^{-1}
	ϵ_{RMSE}	1.2959×10^{-1}	$\mathbf{8.0500 \times 10^{-2}}$	1.4896
	ϵ_{SSE}	3.4661×10^{1}	$\mathbf{1.3654 \times 10^{1}}$	4.8395×10^{3}
	ϵ_{σ}	1.1498×10^{-1}	$\mathbf{7.5150 \times 10^{-2}}$	8.4606×10^{-1}
$Rotation\ (deg)$	ϵ_{max}	$\mathbf{5.1192}$	8.7111	3.2931×10^{1}
	ϵ_{mean}	8.2105×10^{-1}	$\mathbf{4.6259 \times 10^{-1}}$	1.4803×10^{1}
	ϵ_{median}	3.7943×10^{-1}	$\mathbf{5.4208 \times 10^{-2}}$	1.3513×10^{1}
	ϵ_{min}	3.5400×10^{-3}	$\mathbf{1.6990 \times 10^{-3}}$	1.5414
	ϵ_{RMSE}	1.4108	$\mathbf{1.1421}$	1.6918×10^{1}
	ϵ_{SSE}	4.1083×10^{3}	$\mathbf{2.7484 \times 10^{3}}$	6.2428×10^{5}
	ϵ_{σ}	1.1473	$\mathbf{1.0442}$	8.1920

5.3.2 Case B: Different models

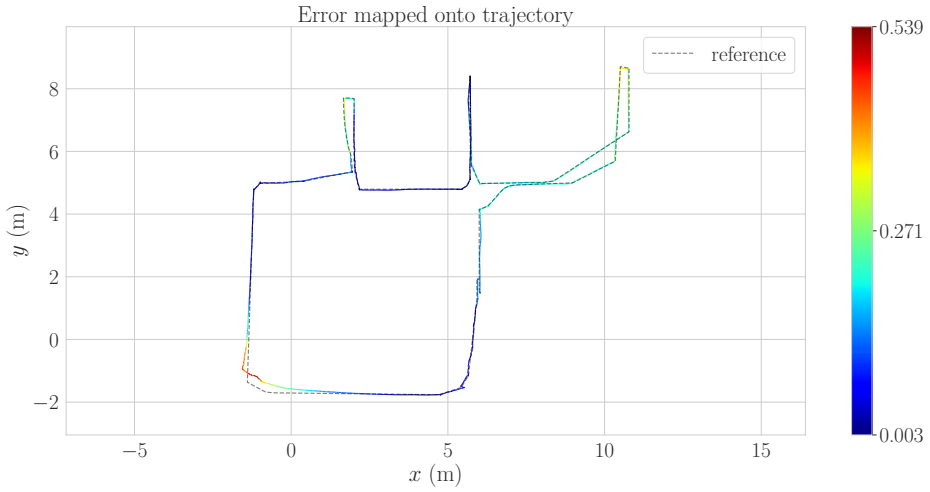
The earlier case is straightforward where the collision and mesh correspondences model are the same. To make it more challenging, fig. 5.1a is used for collisions while fig. 5.1b is used for mesh correspondences. Due to the lack of furniture, there will be outliers as well.

An avid reader might note that the results of standard LOAM are the same as before as shown in fig. 5.3c. This result is expected as changing model for mesh correspondences does not affect LOAM that does not consider mesh feature. A quantitative error analysis is presented in table 5.3.

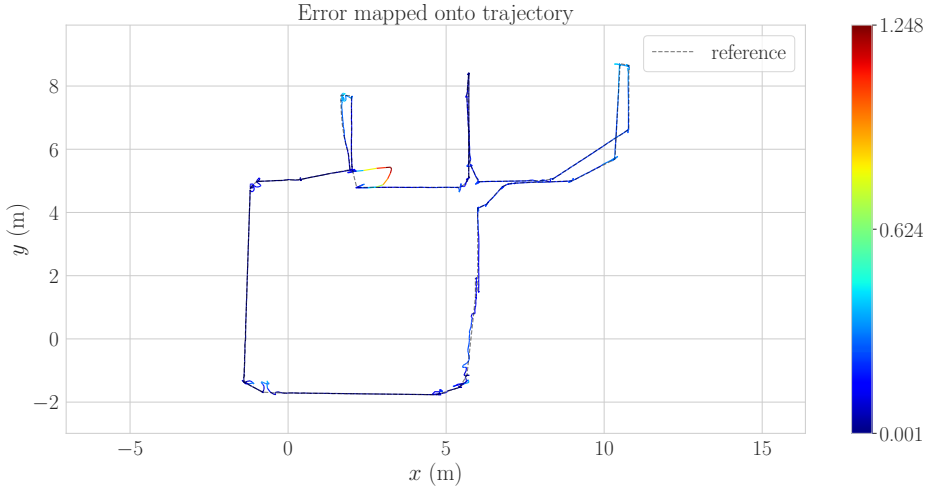
Apart from the minimum and maximum errors, the other metrics are pretty close for MA-LOAM and cluster ICP. Even this environment is not challenging enough as there is not much difference between the two models apart from furniture, which is also scarce. More complicated scenarios are explored in the next section, mainly by increasing the scan model differences and adding dynamic elements.

5.4 Office EG

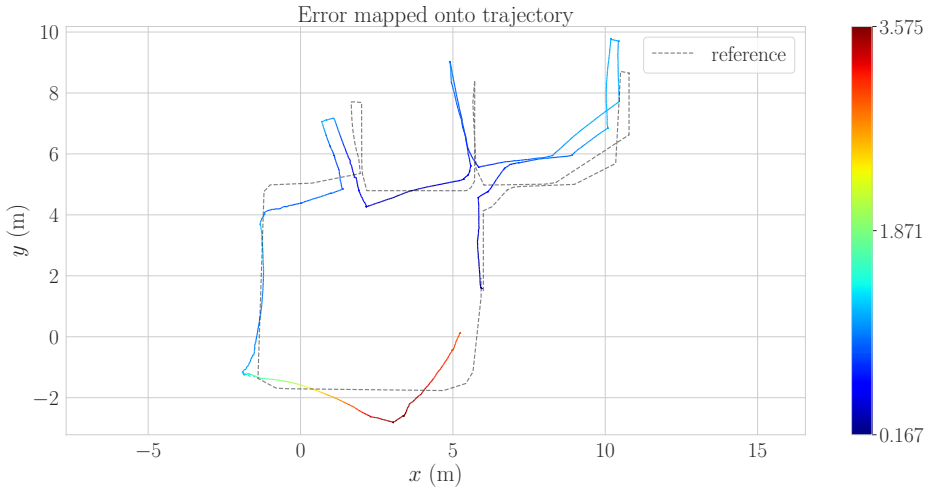
This environment is based on an office building ground floor. The robot trajectory is different for each case; therefore, it is not shown. Figure 5.4a shows the environment model used for mesh correspondences while fig. 5.4b illustrates what the office looks like after putting the furniture. As the transformation between odometry and world frame is unknown, Umeyama alignment is used to compare the results. Different cases are discussed below.



(a) Path estimated by MA-LOAM

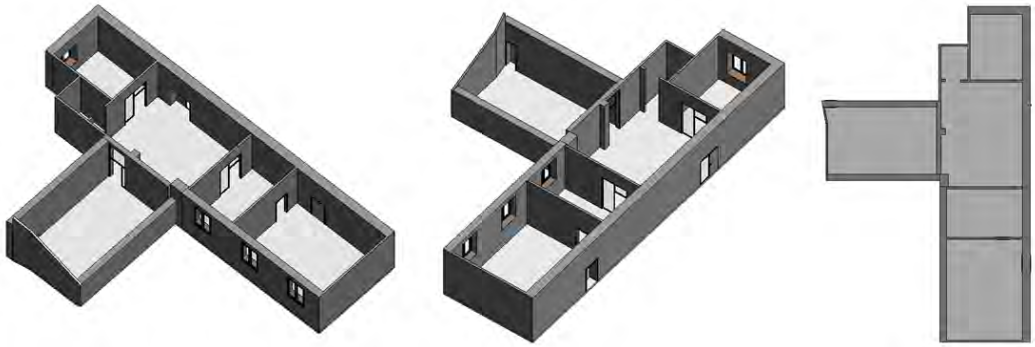


(b) Path estimated by cluster ICP



(c) Path estimated by standard LOAM

Figure 5.3: Haus FZK, Case B: Comparison of the agent path estimated by different algorithms when the collision and mesh correspondences model is different



(a) Empty office model which is used for mesh correspondences. The doors and windows are filtered out and not used for feature matching

(b) Actual office with furniture leading to significant differences with environment in fig. 5.4a

(c) Office after a heavy disaster

Figure 5.4: Office EG: Different environment models for office EG

Table 5.3: Haus FZK, Case B: Error statistics for different algorithms when the collision and mesh correspondences model is different

	Error Metric	MA-LOAM	Cluster ICP	LOAM
$Translation\ (m)$	ϵ_{max}	$\mathbf{5.3914 \times 10^{-1}}$	1.2476	3.5747
	ϵ_{mean}	1.5582×10^{-1}	$\mathbf{1.4841 \times 10^{-1}}$	1.2260
	ϵ_{median}	1.4352×10^{-1}	$\mathbf{1.4345 \times 10^{-1}}$	1.0199
	ϵ_{min}	2.7050×10^{-3}	$\mathbf{5.0400 \times 10^{-4}}$	1.6656×10^{-1}
	ϵ_{RMSE}	$\mathbf{1.9054 \times 10^{-1}}$	1.9189×10^{-1}	1.4896
	ϵ_{SSE}	7.4646×10^{1}	$\mathbf{7.4269 \times 10^{1}}$	4.8395×10^{3}
	ϵ_{σ}	$\mathbf{1.0966 \times 10^{-1}}$	1.2164×10^{-1}	8.4606×10^{-1}
$Rotation\ (deg)$	ϵ_{max}	$\mathbf{7.2590}$	1.8810×10^{1}	3.2931×10^{1}
	ϵ_{mean}	$\mathbf{1.7124}$	1.7706	1.4803×10^{1}
	ϵ_{median}	1.2611	$\mathbf{1.2157}$	1.3513×10^{1}
	ϵ_{min}	1.5882×10^{-2}	$\mathbf{1.1530 \times 10^{-3}}$	1.5414
	ϵ_{RMSE}	$\mathbf{2.1619}$	2.6484	1.6918×10^{1}
	ϵ_{SSE}	$\mathbf{9.6091 \times 10^{3}}$	1.4148×10^{4}	6.2428×10^{5}
	ϵ_{σ}	$\mathbf{1.3196}$	1.9696	8.1920

Case A: Empty office with people

This case is primarily focused on adding dynamic elements to the environment. The collision and mesh correspondences model is the same (fig. 5.4a). In addition, humans are also walking around, adding dynamic noise. The error statistics are presented in table 5.4. MA-LOAM performs reasonably well compared to the other two approaches. As there are no scan model differences, the estimated trajectory of cluster ICP and MA-LOAM are quite close to each other.

Table 5.4: Office EG, Case A: Error statistics for different algorithms when the collision and reference model are the same. Additionally, humans are also walking around to add dynamic elements to the environment

	Error Metric	MA-LOAM	Cluster ICP	LOAM
$Translation\ (m)$	ϵ_{max}	$\mathbf{1.0384 \times 10^{-1}}$	1.7403×10^{-1}	1.1292×10^{1}
	ϵ_{mean}	$\mathbf{2.4164 \times 10^{-2}}$	2.6378×10^{-2}	2.7997
	ϵ_{median}	$\mathbf{1.9869 \times 10^{-2}}$	2.0447×10^{-2}	1.8646
	ϵ_{min}	$\mathbf{1.6610 \times 10^{-3}}$	2.2520×10^{-3}	6.0991×10^{-2}
	ϵ_{RMSE}	$\mathbf{2.8346 \times 10^{-2}}$	3.2922×10^{-2}	3.7740
	ϵ_{SSE}	$\mathbf{1.6415}$	2.2143	2.9098×10^{4}
	ϵ_{σ}	$\mathbf{1.4819 \times 10^{-2}}$	1.9700×10^{-2}	2.5308
$Rotation\ (deg)$	ϵ_{max}	$\mathbf{4.2922}$	5.3872	7.0657×10^{1}
	ϵ_{mean}	6.2659×10^{-1}	$\mathbf{6.2347 \times 10^{-1}}$	3.2584×10^{1}
	ϵ_{median}	$\mathbf{4.4893 \times 10^{-1}}$	4.5178×10^{-1}	2.4573×10^{1}
	ϵ_{min}	$\mathbf{1.0578 \times 10^{-2}}$	1.6155×10^{-2}	7.6808
	ϵ_{RMSE}	8.4693×10^{-1}	$\mathbf{8.3714 \times 10^{-1}}$	3.6960×10^{1}
	ϵ_{SSE}	1.4654×10^{3}	$\mathbf{1.4317 \times 10^{3}}$	2.7908×10^{6}
	ϵ_{σ}	5.6980×10^{-1}	$\mathbf{5.5865 \times 10^{-1}}$	1.7445×10^{1}

5.4.1 Case B: Furnished office without people

This scenario focuses on significant scan model differences. For the LiDAR collisions, fig. 5.4b is used while for the mesh correspondences, fig. 5.4a is used. Similar to previous cases, the performance is evaluated for MA-LOAM, cluster ICP and LOAM.

With the increase in scan model differences, the proposed approach, namely MA-LOAM performs better. As shown in fig. 5.5b, the estimated path by cluster ICP is quite rough and oscillatory compared to the computed trajectory of MA-LOAM in fig. 5.5a. Table 5.5 shows errors compared to the ground truth using Umeyama alignment.

Table 5.5: Office EG, Case B: Error statistics for different algorithms when the actual office is furnished while the reference model is empty

	Error Metric	MA-LOAM	Cluster ICP	LOAM
$Translation\ (m)$	ϵ_{max}	$\mathbf{4.0650 \times 10^{-1}}$	5.4152×10^{-1}	1.8398
	ϵ_{mean}	$\mathbf{1.0304 \times 10^{-1}}$	1.0684×10^{-1}	6.7989×10^{-1}
	ϵ_{median}	8.1748×10^{-2}	$\mathbf{8.0983 \times 10^{-2}}$	6.7188×10^{-1}
	ϵ_{min}	1.8232×10^{-2}	$\mathbf{1.0430 \times 10^{-2}}$	1.9383×10^{-1}
	ϵ_{RMSE}	$\mathbf{1.2204 \times 10^{-1}}$	1.2845×10^{-1}	7.3924×10^{-1}
	ϵ_{SSE}	$\mathbf{3.1127 \times 10^{1}}$	3.4484×10^{1}	1.1421×10^{3}
	ϵ_{σ}	$\mathbf{6.5396 \times 10^{-2}}$	7.1306×10^{-2}	2.9021×10^{-1}
$Rotation\ (deg)$	ϵ_{max}	**8.1035**	1.4002×10^{1}	3.3953×10^{1}
	ϵ_{mean}	**1.4361**	1.5278	1.7376×10^{1}
	ϵ_{median}	**1.1152**	1.1699	1.7040×10^{1}
	ϵ_{min}	$\mathbf{4.5200 \times 10^{-2}}$	6.5393×10^{-2}	4.1859
	ϵ_{RMSE}	**1.7890**	1.9872	1.8794×10^{1}
	ϵ_{SSE}	$\mathbf{6.6892 \times 10^{3}}$	8.2533×10^{3}	7.3820×10^{5}
	ϵ_{σ}	**1.0668**	1.2708	7.1621

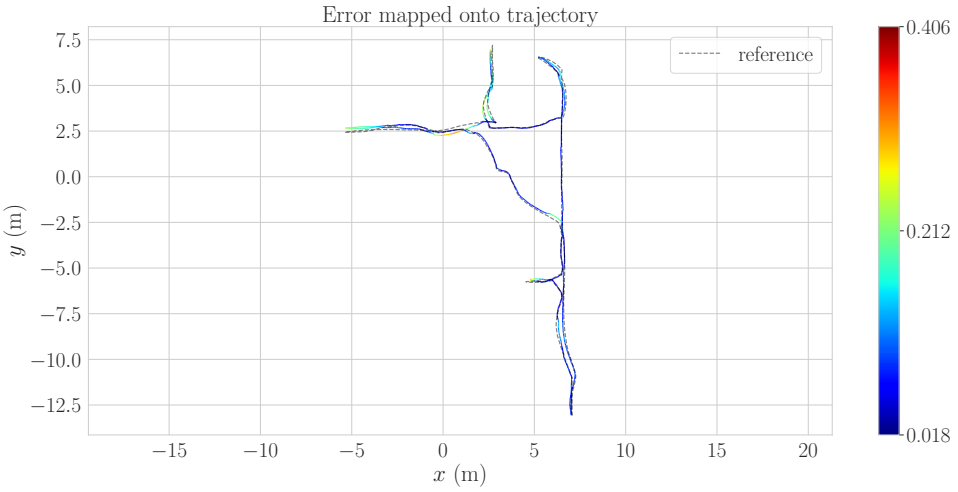
The error metrics favor the proposed methodology and estimate paths closer to the ground truth. The estimated trajectory also directly affects the mapping part of the SLAM algorithm, which for this case is LOAM, though that is not the focus of this thesis.

5.4.2 Case C: Office after a disaster without people

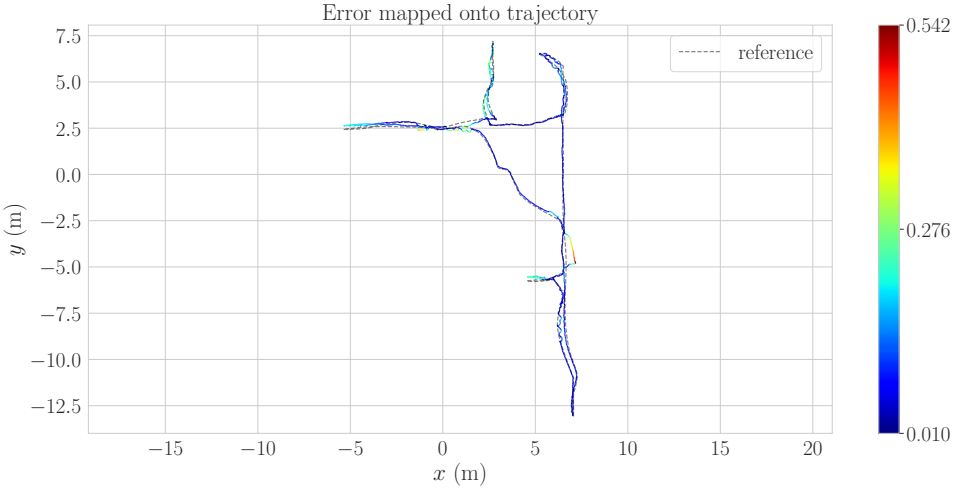
This instance is pivoted around adding heavy noise to the collision model. After experiencing a disaster, the office contains numerous objects not present in the model used for mesh correspondences. Specifically, fig. 5.4c is used for collisions while fig. 5.4a is utilized for mesh correspondences. Table 5.6 presents the translational and rotational errors of the computed trajectories for each algorithm.

5.4.3 Case D: Office after a disaster with people

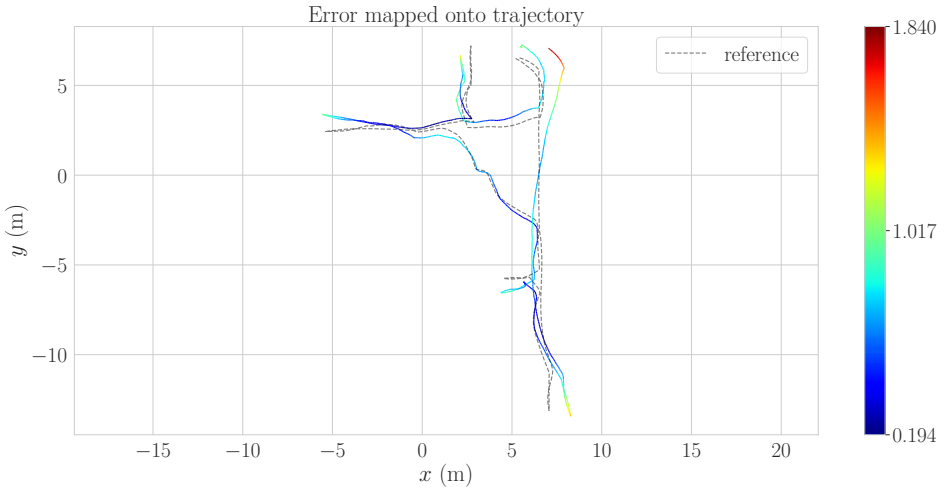
This scenario focuses on where the office building experienced a heavy disaster leading to broken furniture and other obstacles. Also, humans are simulated to add dynamic



(a) Path estimated by MA-LOAM



(b) Path estimated by cluster ICP



(c) Path estimated by standard LOAM

Figure 5.5: Office EG, Case B: Comparison of the agent path estimated by different algorithms when the actual office is furnished while the reference model is empty

Table 5.6: Office EG, Case C: Error statistics for different algorithms when the office experienced a heavy disaster with no dynamic elements

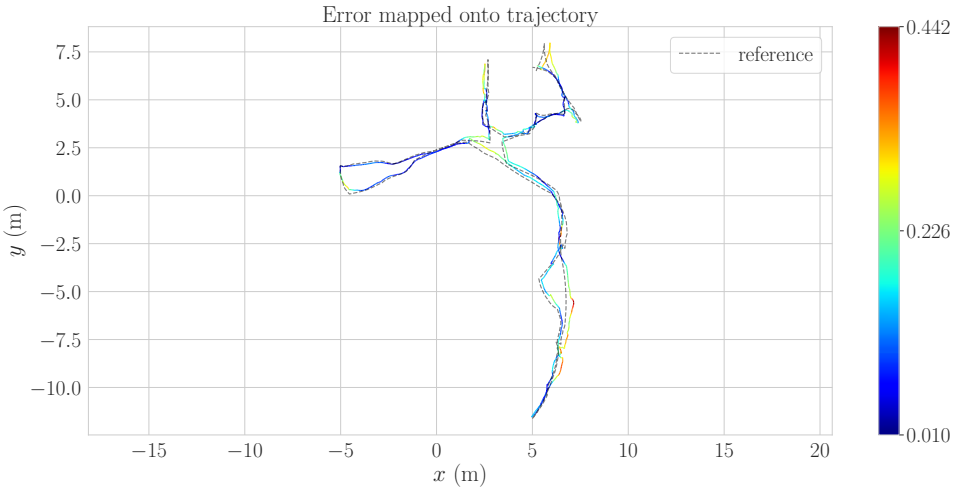
	Error Metric	MA-LOAM	Cluster ICP	LOAM
$Translation\ (m)$	ϵ_{max}	$\mathbf{4.1128 \times 10^{-1}}$	9.7541×10^{-1}	4.0452
	ϵ_{mean}	$\mathbf{1.3666 \times 10^{-1}}$	1.9965×10^{-1}	1.6032
	ϵ_{median}	$\mathbf{1.2988 \times 10^{-1}}$	1.4740×10^{-1}	1.2784
	ϵ_{min}	$\mathbf{7.6810 \times 10^{-3}}$	1.0346×10^{-2}	4.8978×10^{-1}
	ϵ_{RMSE}	$\mathbf{1.5186 \times 10^{-1}}$	2.6079×10^{-1}	1.9003
	ϵ_{SSE}	$\mathbf{6.2500 \times 10^{1}}$	1.8431×10^{2}	9.7865×10^{3}
	ϵ_{σ}	$\mathbf{6.6239 \times 10^{-2}}$	1.6779×10^{-1}	1.0203
$Rotation\ (deg)$	ϵ_{max}	$\mathbf{1.8692 \times 10^{1}}$	3.4266×10^{1}	4.1218×10^{1}
	ϵ_{mean}	**3.3606**	4.3719	2.3818×10^{1}
	ϵ_{median}	**2.5363**	2.7548	2.3605×10^{1}
	ϵ_{min}	$\mathbf{2.0239 \times 10^{-1}}$	3.2532×10^{-1}	4.3415
	ϵ_{RMSE}	**4.2352**	5.9046	2.5624×10^{1}
	ϵ_{SSE}	$\mathbf{4.861 \times 10^{4}}$	9.4482×10^{4}	1.7793×10^{6}
	ϵ_{σ}	**2.5775**	3.9687	9.4478

elements to the environment. Both these factors lead to increased complexity for solving the SLAM problem. Once again, for mesh correspondences, model in fig. 5.4a is used. For the LiDAR collision, fig. 5.4c is utilized for scans.

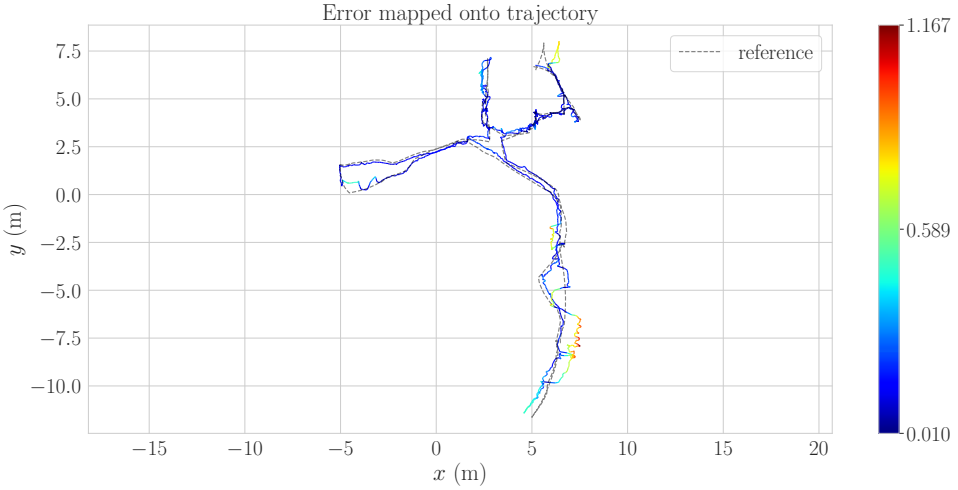
With the increased complexity of the given scenario, the advantage of the proposed methodology is self-evident. In a dynamic environment with significant differences between the collision and mesh correspondences model, MA-LOAM outperforms the other two approaches. This is also validated by the statistics in table 5.7.

Table 5.7: Office EG, Case D: Error statistics for different algorithms when the office experienced a heavy disaster and people are walking

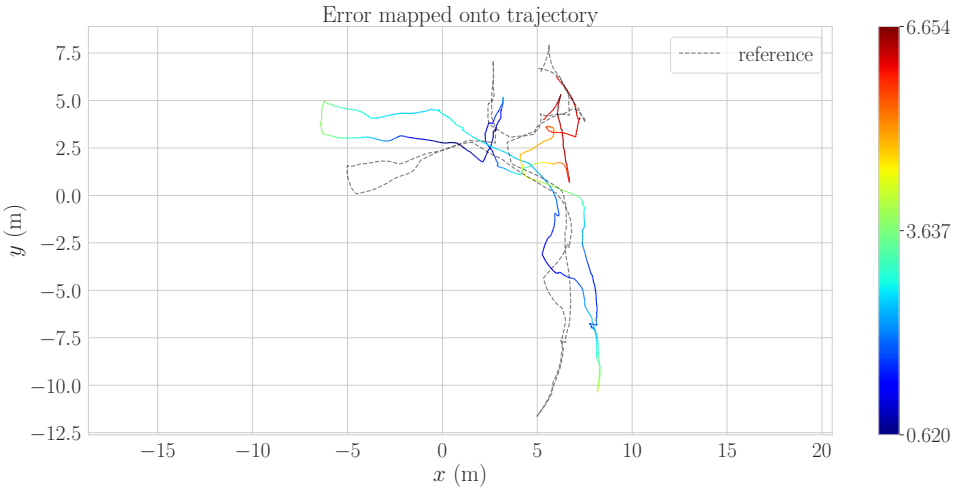
	Error Metric	MA-LOAM	Cluster ICP	LOAM
$Translation\ (m)$	ϵ_{max}	$\mathbf{4.4241 \times 10^{-1}}$	1.1671	6.6541
	ϵ_{mean}	$\mathbf{1.5684 \times 10^{-1}}$	2.4801×10^{-1}	3.1568
	ϵ_{median}	$\mathbf{1.3766 \times 10^{-1}}$	1.6296×10^{-1}	2.7186
	ϵ_{min}	$\mathbf{9.7460 \times 10^{-3}}$	1.0186×10^{-2}	6.1980×10^{-1}
	ϵ_{RMSE}	$\mathbf{1.8121 \times 10^{-1}}$	3.3269×10^{-1}	3.5940
	ϵ_{SSE}	$\mathbf{8.0417 \times 10^{1}}$	2.7106×10^{2}	3.1634×10^{4}
	ϵ_{σ}	$\mathbf{9.0766 \times 10^{-2}}$	2.2175×10^{-1}	1.7181
$Rotation\ (deg)$	ϵ_{max}	$\mathbf{2.0050 \times 10^{1}}$	3.7562×10^{1}	5.1889×10^{1}
	ϵ_{mean}	**4.5449**	5.3570	3.1333×10^{1}
	ϵ_{median}	**3.7739**	3.9197	3.0081×10^{1}
	ϵ_{min}	3.2762×10^{-1}	$\mathbf{2.4966 \times 10^{-1}}$	6.0391
	ϵ_{RMSE}	**5.5577**	7.4653	3.2698×10^{1}
	ϵ_{SSE}	$\mathbf{7.5646 \times 10^{4}}$	1.3649×10^{5}	2.6184×10^{6}
	ϵ_{σ}	**3.1988**	5.1994	9.3501



(a) Path estimated by MA-LOAM



(b) Path estimated by cluster ICP



(c) Path estimated by standard LOAM

Figure 5.6: Office EG, Case D: Comparison of the agent path estimated by different algorithms

Looking at table 5.7, the performance of the proposed MA-LOAM remains consistent even for complicated situations. The scan model differences and dynamic elements have adverse effects on the other two algorithms.

5.5 Mapping

So far, the results have been presented in detail for the odometry part, as the main contribution of this thesis is in that direction. Nonetheless, the mapping result of one of the cases (section 5.4.2) is shown just for completeness. For mapping, the path estimated by the odometry module is further optimized, and scans are stitched together to create the final map of the environment.

(a) Mapping result of MA-LOAM

(b) Mapping result of LOAM

Figure 5.7: Sample of map generated by MA-LOAM and LOAM

Figure 5.7 shows the computed maps by two different approaches. The maps are color-coded using their height. The estimated maps should correspond to fig. 5.4c. Ideally, improving odometry estimates should also increase the mapping accuracy. To compare the map with the ground truth, the computed map is aligned with the environment model using the standard ICP algorithm. Then, the average distance of the point cloud to the underlying environment model is computed. Notice that the cluster ICP has no mapping part; therefore, its estimated map is not shown. MA-LOAM performs marginally better in mapping as shown in table 5.8.

Table 5.8: Mapping accuracy of different approaches: The generated map is aligned to the environment model using standard ICP with an overlap of 90%

Max Distance (m)	Error Metric (m)	MA-LOAM	LOAM
0.1	ϵ_{mean}	$\mathbf{1.1298 \times 10^{-2}}$	1.3464×10^{-2}
	ϵ_{σ}	4.3421×10^{-2}	$\mathbf{4.1858 \times 10^{-2}}$
∞	ϵ_{mean}	$\mathbf{4.9799 \times 10^{-1}}$	4.9863×10^{-1}
	ϵ_{σ}	**1.9246**	1.9290

5.6 Global Registration

The final element of the proposed pipeline is global registration. As discussed earlier, a reasonable initial pose estimate is required to compute the mesh features. In a case where the initial pose is far from the ground truth, the proposed methodology fails. This is shown in fig. 5.8. Of course, LOAM which does not depend upon any mesh features, is not influenced by this wrong pose estimate.

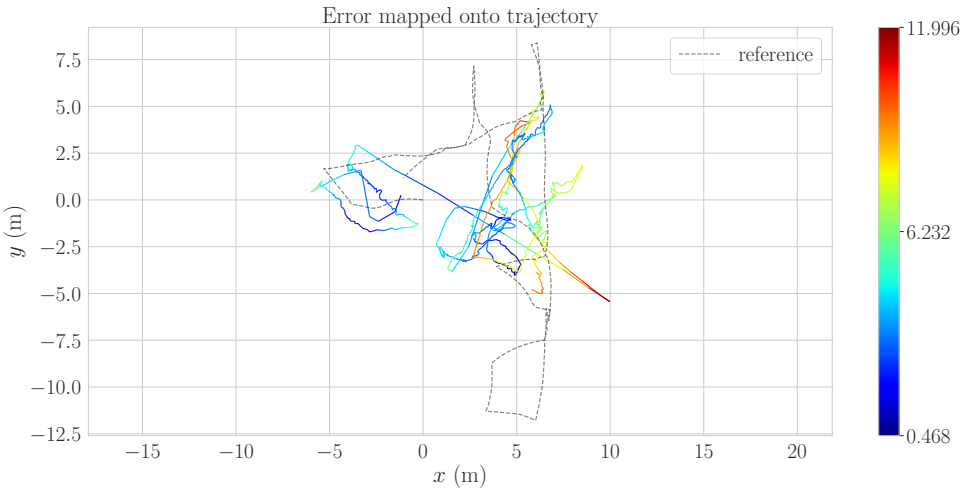


Figure 5.8: Failure of MA-LOAM due to wrong initial pose estimate

For this purpose, different global registration techniques are explored. Unsurprisingly, traditional approaches usually fail to produce a reasonable estimate in cases of partial overlap. The case of DGR is more nuanced. While DGR is able to produce a reasonable

pose estimate for the Haus FZK cases (section 5.3), it fails for the Office EG cases (section 5.4). The underlying model used for this purpose is trained on the KITTI data provided in the official implementation. Furthermore, the model is not retrained and is used as-is. A quantitative analysis of different approaches for Haus FZK is depicted in table 5.9.

Table 5.9: Global registration using different algorithms

	Ground Truth	DGR[a]	FGR[b]	RANSAC[b]
m	6.0000	6.0614	5.9426	9.0468
	1.5000	1.6082	4.0503	3.1602
	5.8800×10^{-1}	4.7943×10^{-1}	1.4725	3.3565
	ϵ_{transl}	$\mathbf{1.6515 \times 10^{-1}}$	2.6999	4.4389
deg	0	1.2570	0	-1.6495×10^{2}
	0	-9.5035×10^{-1}	0	1.1508×10^{1}
	0	3.4026×10^{-1}	0	-4.9065×10^{1}
	ϵ_{rot}	1.6143	**0**	1.6161×10^{2}

[a] With FCGF and voxel size of 30 cm
[b] Using Fast Point Feature Histogram (FPFH)

Note that the RANSAC algorithm is based on random selection and can produce different results given the same input. From table 5.9, it is evident that DGR is able to generalize to unknown environments without the need for retraining. The pose estimate is close to the ground truth but still has a relatively high error. In order to refine the estimate, cluster ICP can be used. Nonetheless, it should be used with caution as it may fail and can lead to completely unusable results as shown in fig. 5.8.

5.7 Computational Time Analysis

There are a lot of hyperparameters that can affect the computational performance of MA-LOAM. This section limits itself to just the voxel size as it is the primary parameter determining the total number of features. The number of features per voxel is limited to a maximum of 3. Table 5.10 shows the time required to solve the complete odometry problem for a particular timestamp.

Table 5.10: Average computational time required for computing the mesh features and solving odometry at a single timestamp. Decreasing voxel size increases the number of features thus leading to more time consumed for odometry. As a reference, for the same scenario, $|\mathcal{F}_{\mathcal{E}}| \approx 180$ and $|\mathcal{F}_{\mathcal{S}}| \approx 300$

Voxel Size (m)	$\sim \lvert\mathcal{F}_{\mathcal{M}}\rvert$	$\sim \mathcal{T}_{\mathcal{F}_{\mathcal{M}}}(ms)$	$\sim \mathcal{T}_{total}(ms)$
1	350	20	90
0.5	990	16	120
0.25	2600	14	190
0.1	4700	13	350

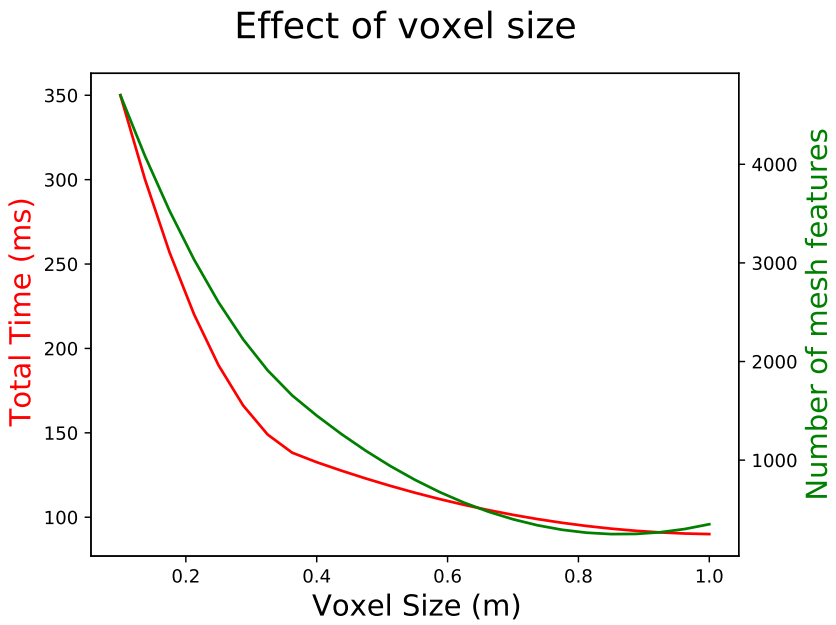


Figure 5.9: Effect of voxel size on computational time and the number of mesh features

When plotted onto a graph and quadratically interpolated in between the points, fig. 5.9 is obtained. As expected, as the voxel size increases, the number of retained mesh features decreases, leading to a reduction in computational time. If the confidence in the environment model is high, i.e., when the reality is close to the reference model, the weight of the mesh features $\lambda_{\mathcal{M}}$ can be increased in eq. (4.9) as the voxel size is increased. Larger voxels should result in more prominent features being retained while the fine detail is lost. The selection of only large features might be beneficial to reduce the effect of random noise.

Chapter 6

Conclusion And Outlook

This section wraps up and goes over the key findings of this thesis, along with their limitations. Furthermore, some proposals for future work are presented.

6.1 Findings

This thesis mainly dealt with robustly integrating the environment model into the LOAM framework. In the beginning, in section 1.5, particular questions were posed to define the outline more concretely. They shall now be addressed explicitly here:

1. *What useful information can be extracted from the BIM model?*
   BIM model, contrary to the standard CAD model, contains semantic information as well. This extra level of information allows for specific elements to be filtered out. For example, dynamic elements such as doors and windows might be discarded as they do not add valuable information for feature matching. Similarly, elements that are not visible to the sensor, such as glass, can also be filtered out. The BIM model can also contain temporal information, in which case a time-dependent environment model can be obtained. Otherwise, the entire environment model can be used.

   There are two ways to extract the information from this environment model. A common method is using Poisson sampling and discretizing the environment given the maximum number of sampling points. The other method avoids this discretization and directly uses the geometric information available. In this thesis, both forms are used. When computing the mesh features for MA-LOAM, the environment model is not discretized but instead used directly to find the closest point on the mesh. Conversely, when using DGR for the initial global pose estimate, the environment is sampled using the said Poisson distribution. The main reason is that the underlying neural network of DGR expects two point clouds and not a continuous model of the environment.

2. *How can the extracted information be integrated into LOAM algorithm?*
   The extracted mesh features from the LiDAR along with the environment model (stored inside a BVH along with the corresponding k-D tree) are used to find point-to-mesh correspondences pairs. These pairs are then formulated as robust distance loss and added to the cost function as already discussed in eq. (4.6) and eq. (4.9). These mesh features are added directly to the odometry module. Better odometry estimates lead to better map quality, thus improving both.

3. *In what ways can the effect of noise/outliers be minimized? How does it stack up against the base algorithm?*
   The effect of outliers on the joint optimization can be adverse. In order to mitigate this, special care is taken to reduce the outlier's effect. For example, when clustering the points in section 4.2.2, if the cluster size is smaller than a threshold, the cluster is not considered valid and therefore dropped. In the joint optimization (eq. (4.9)), robust loss functions are used to reduce the effect of outliers. Also, if the distance between the correspondence pair exceeds a certain threshold for any features, it is removed. These techniques combined seem to tolerate outliers reasonably well.

4. *Is it possible to run the methodology in real-time? What part of the algorithm contributes the most to execution time?*
   Addressing this objective is more complex. As already shown in section 5.7, the execution time is dependent upon many hyperparameters. Limiting the discussion to only the voxel size, it is possible to run the odometry module at $10\,\mathrm{Hz}$ if a large voxel size is selected. The downside is that a large voxel size usually leads to lower odometry accuracy. To limit the mesh features, perhaps a fixed number of features can be selected from the computed feature set $\mathcal{F}_{\mathcal{M}}$ based on a fitness criterion such as giving importance to features that originated from large clusters or point descriptors. Usually, most of the time is spent solving the optimization problem.

5. *Can the approach generalize to environments where the scan differs significantly from the model?*
   The approach can adapt to scan model differences as shown in figs. 5.3a, 5.5a and 5.6a. Nonetheless, it still assumes that the scale of the environment model is similar to the actual building. The minimization problem does not correct or account for scale differences. Also, differences that are smaller in scale, like a scaffold, are easy to reject. However, bigger differences like false ceilings and partition walls not in the design model are harder to handle. Nonetheless, the distance-based rejection method proposed in this thesis explicitly deals with such a scenario.

6.2 Contributions

The work in this thesis integrated the BIM model into SLAM computations. The salient contributions are as follows:

- A robust strategy is proposed to extract mesh features from a dense point cloud using voxel grids and an unsupervised clustering algorithm. This approach preserves the finer details in the point cloud while removing redundant and unreliable points. The methodology is similar to that of Tazir et al., 2018. The main difference is that it is used as a pre-processing step to extract the mesh features and not for matching two point clouds.

- An approach for integrating the mesh features into the LOAM framework is presented. The cost function is modeled as a joint optimization problem that considers the corner, surface, and mesh features. This joint optimization is similar to OELSCH et al., 2021 but differs significantly in the computation of mesh features and the weighting strategy used for the optimization.
- To limit the maximum number of features used, a selection strategy is proposed based on the cluster size and their distance from the sensor. This method allows to speed up the optimization as there are fewer features while also reducing the number of outliers.
- As the proposed approach requires a reasonable estimate of the initial pose in the global frame, DGR CHOY et al., 2020 is employed with minor changes and integrated into the odometry pipeline.
- The full implementation of this thesis is open-source and available publicly[1].

6.3 Outlook

There are many areas of improvement that the proposed approach can benefit from. As the list of improvements is rather exhaustive, only some of the crucial improvements are mentioned.

6.3.1 Loop closure

Currently, there is no loop closure technique employed to detect revisits. Currently, most of the famous SLAM frameworks are graph-based and utilize loop closure (such as google-cartographer, ORB-SLAM, LSD-SLAM) to reduce errors. Each loop closure detection allows adding an extra constraint between the poses in the detected loop. This loop closure reduces the drift in the long term, thus allowing better odometry estimates. This feature is particularly useful in cases where only a partial or no environment model is available.

6.3.2 Optimal Feature Weights

In the proposed approach, a feature weighting is suggested in eq. (4.9). For most cases, all the Lagrangian multipliers are set to 1. A better weighting strategy can be developed. At the very least, a comparative study can reveal what values work best in different scenarios. Then, this process can be automated to change weights on the fly.

[1] https://github.com/darkscyla/MA-LOAM

6.3.3 Fitness Strategy for Mesh Features

As already discussed (table 5.10), the number of mesh features is usually way higher than the counterpart edge and surface features. A fitness strategy can be developed to select only the high fitness score points. To this end, standard point descriptors such as SIFT, SURF, or ORB might be used. This scoring will come with two significant benefits. The first one, of course, is allowing only the use of a subset of points making the computations way faster. The second benefit is having another outlier rejection step, as a low fitness score will indicate a poor feature.

6.3.4 Model Preprocessing Stage

The methodology proposed extracts the environment from the BIM model and uses it for point-to-mesh feature correspondences. This process can be significantly sped up without loss in accuracy if the environment model can be simplified while retaining the essential details. More often than not, the tiny details in the environment model do not add any extra value as the LiDAR sensor has limited sensitivity. For example, a sub-millimeter feature observed even from a relatively near position will not appear as more than a single point. Retaining such a feature only adds to the computational complexity and does not improve localization accuracy. A suitable computational geometry algorithm such as vertex clustering [Low and Tan, 1997] or incremental decimation [Shekhar et al., 1996, Hoppe, 1996, Garland and Heckbert, 1997] can be employed and investigated to improve the computational time.

6.4 Conclusion

This thesis proposed a robust methodology to integrate the BIM model into the LOAM framework. In addition to the standard edge and surface features of LOAM, mesh features are added, and correspondences are found using the underlying environment model. Special care is taken to remove and reduce the effect of outliers on the whole optimization. One of the main limitations of the proposed methodology is a reasonable initial global pose estimate. In a case where the estimated pose is far from reality, the optimization fails and cannot recover in most cases.

While the proposed algorithm can be improved in many areas, the enhancement in the accuracy is still quite notable. By utilizing the extra level of information, i.e., the environment model, odometry can be better estimated. This enhancement, in turn, improves cartography as well. Special attention is given to utilizing appropriate data structures to avoid costly lookups. As the mesh features require a reasonable global pose estimate, an appropriate deep learning methodology DCP is selected among other alternatives based on its performance and ability to generalize to new environments.

Using the proposed methodology offers multiple advantages. Most modern buildings have a corresponding CAD or BIM model, meaning that no additional cost or effort is required for acquiring the building plan. Therefore, odometry estimates can be improved without any charge. Furthermore, for some cases, additional sensors may be placed to accurately determine the agent's position depending on the localization requirements. In such cases, the proposed approach can supplement the current pose estimate, thus requiring fewer or potentially no additional sensors at all. The methodology proposed reduces the overall drift for the agent's trajectory. MA-LOAM also seems to perform reasonably well in an environment with dynamic elements. Scan and mesh correspondences model differences are also tolerated pretty well. As most environment models do not contain furniture and multiple other details, the mesh correspondence model will always be different from reality. The ability to deal with such differences allows the proposed approach to work in a variety of scenarios. Lastly, the proposed methodology is entirely open source. This should allow interested individuals to quickly try out the methodology and evaluate the fitness for their use case.

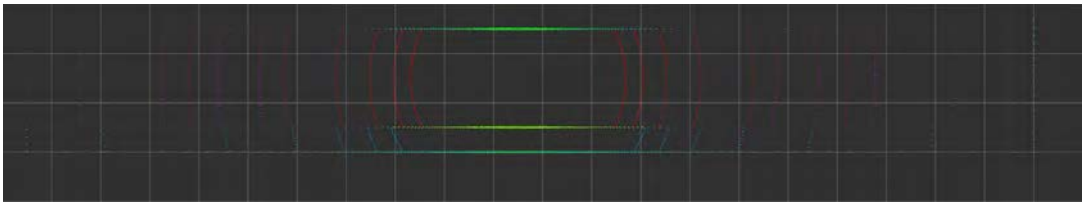
Appendix A

Hardware / Setup

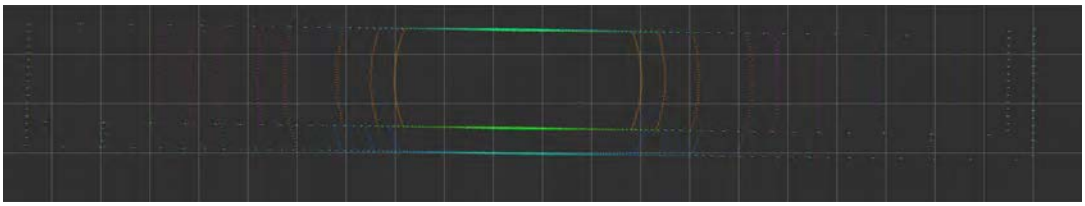
The proposed approach is tested on the following machine:

- **CPU:** Intel Core i7 9750H
- **RAM:** 32 GB
- **GPU:** RTX 2060
- **OS:** Linux 20.04 running on WSL Windows 11

GPU was initially used to simulate the LiDAR scans, but due to a known bug in Gazebo simulator[1], it is was dropped and is no longer used for any computations. This problem is shown in fig. A.1. The optimization problem is solved solely on the CPU as well. The proposed algorithm should perform better in cases where simultaneously running the simulation is not necessary, for example, a real LiDAR scanner or pre-recorded data.



(a) LiDAR scan computed using CPU



(b) LiDAR scan computed using GPU

Figure A.1: CPU and GPU return different results for the same scan. The CPU version is actually correct while the GPU version is distorted as the underlying environment is axis aligned

[1] See here

Bibliography

AJANKI, A. (2007). *Example of k-nearest neighbour classification*. https://upload.wikimedia.org/wikipedia/commons/e/e7/KnnClassification.svg

ARMENI, I., SENER, O., ZAMIR, A. R., JIANG, H., BRILAKIS, I., FISCHER, M., & SAVARESE, S. (2016). 3d semantic parsing of large-scale indoor spaces. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1534–1543.

ARUN, K. S., HUANG, T. S., & BLOSTEIN, S. D. (1987). Least-squares fitting of two 3-d point sets. *IEEE Transactions on pattern analysis and machine intelligence*, (5), 698–700.

BARZILAI, J., & BORWEIN, J. M. (1988). Two-point step size gradient methods. *IMA journal of numerical analysis*, *8*(1), 141–148.

BAY, H., TUYTELAARS, T., & VAN GOOL, L. (2006). Surf: Speeded up robust features. *European conference on computer vision*, 404–417.

BELAGIANNIS, V., RUPPRECHT, C., CARNEIRO, G., & NAVAB, N. (2015). Robust optimization for deep regression. *Proceedings of the IEEE international conference on computer vision*, 2830–2838.

BENTLEY, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, *18*(9), 509–517.

BERCHTOLD, S., BÖHM, C., & KRIEGAL, H.-P. (1998). The pyramid-technique: Towards breaking the curse of dimensionality. *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, 142–153.

BERLAND. (2007a). *Forward accumulation in automatic differenciation*. https://en.wikipedia.org/w/index.php?title=File:ForwardAccumulationAutomaticDifferentiation.png

BERLAND. (2007b). *Reverse accumulation in automatic differenciation*. https://commons.wikimedia.org/wiki/File:ReverseaccumulationAD.png

BHATLA, A., CHOE, S. Y., FIERRO, O., & LEITE, F. (2012). Evaluation of accuracy of as-built 3d modeling from photos taken by handheld digital cameras. *Automation in construction*, *28*, 116–127.

BLUM, H., STIEFEL, J., CADENA, C., SIEGWART, R., & GAWEL, A. (2020). Precise robot localization in architectural 3d plans. *arXiv preprint arXiv:2006.05137*.

BLUM, M., FLOYD, R. W., PRATT, V. R., RIVEST, R. L., TARJAN, R. E., et al. (1973). Time bounds for selection. *J. Comput. Syst. Sci.*, *7*(4), 448–461.

BONIARDI, F., CASELITZ, T., KÜMMERLE, R., & BURGARD, W. (2019). A pose graph-based localization system for long-term navigation in cad floor plans. *Robotics and Autonomous Systems*, *112*, 84–97.

BURGARD, W., FOX, D., HENNIG, D., & SCHMIDT, T. (1996). Estimating the absolute position of a mobile robot using position probability grids. *Proceedings of the national conference on artificial intelligence*, 896–901.

BURKOV, A. (2019). *The hundred-page machine learning book* (Vol. 1). Andriy Burkov Canada.

CHARBONNIER, P., BLANC-FÉRAUD, L., AUBERT, G., & BARLAUD, M. (1997). Deterministic edge-preserving regularization in computed imaging. *IEEE Transactions on image processing*, *6*(2), 298–311.

CHEN, C., WANG, B., LU, C. X., TRIGONI, N., & MARKHAM, A. (2020). A survey on deep learning for localization and mapping: Towards the age of spatial machine intelligence. *arXiv preprint arXiv:2006.12567*.

CHEN, Y., & MEDIONI, G. (1992). Object modelling by registration of multiple range images. *Image and vision computing*, *10*(3), 145–155.

CHOI, M., SAKTHIVEL, R., & CHUNG, W. K. (2007). Neural network-aided extended kalman filter for slam problem. *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 1686–1690.

CHOY, C., DONG, W., & KOLTUN, V. (2020). Deep global registration. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2514–2523.

CHOY, C., PARK, J., & KOLTUN, V. (2019). Fully convolutional geometric features. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 8958–8966.

CURRAN, K., FUREY, E., LUNNEY, T. F., SANTOS, J. A., WOODS, D., & MCCAUGHEY, A. (2011). An evaluation of indoor location determination technologies. *Journal of Location Based Services*, *5*, 61–78.

DORIGO, M., BIRATTARI, M., & BRAMBILLA, M. (2014). Swarm robotics [revision #138643]. *Scholarpedia*, *9*(1), 1463. https://doi.org/10.4249/scholarpedia.1463

DURRANT-WHYTE, H., & BAILEY, T. (2006). Simultaneous localization and mapping: Part i. *IEEE Robotics Automation Magazine*, *13*(2), 99–110. https://doi.org/10.1109/MRA.2006.1638022

DURRANT-WHYTE, H. F., RYE, D. C., & NEBOT, E. M. (1996). Localization of autonomous guided vehicles.

ELFES, A. (1987). Sonar-based real-world mapping and navigation. *IEEE Journal on Robotics and Automation*, *3*(3), 249–265.

ELFES, A. (1989). Using occupancy grids for mobile robot perception and navigation. *Computer*, *22*(6), 46–57.

EUSTICE, R., WALTER, M., & LEONARD, J. (2005). Sparse extended information filters: Insights into sparsification. *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3281–3288.

FOLLINI, C., MAGNAGO, V., FREITAG, K., TERZER, M., MARCHER, C., RIEDL, M., GIUSTI, A., & MATT, D. T. (2021). Bim-integrated collaborative robotics for application in building construction and maintenance. *Robotics*, *10*(1), 2.

FOX, D., THRUN, S., BURGARD, W., & DELLAERT, F. (2001). Particle filters for mobile robot localization. *Sequential monte carlo methods in practice* (pp. 401–428). Springer.

GARLAND, M., & HECKBERT, P. S. (1997). Surface simplification using quadric error metrics. *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 209–216.

GÉRON, A. (2019). *Hands-on machine learning with scikit-learn, keras, and tensorflow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media.

GOODFELLOW, I., BENGIO, Y., & COURVILLE, A. (2016). *Deep learning*. MIT press.

GRISETTI, G., STACHNISS, C., & BURGARD, W. (2007). Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE transactions on Robotics*, *23*(1), 34–46.

GRUPP, M. (2017). Evo: Python package for the evaluation of odometry and slam.

HADAMARD, J. (1908). *Mémoire sur le problème d'analyse relatif à l'équilibre des plaques élastiques encastrées* (Vol. 33). Imprimerie nationale.

HAVRAN, V. (2000). *Heuristic ray shooting algorithms* (Doctoral dissertation). Ph. d. thesis, Department of Computer Science and Engineering, Faculty of ...

HESS, W., KOHLER, D., RAPP, H., & ANDOR, D. (2016). Real-time loop closure in 2d lidar slam. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 1271–1278.

HOLNESS, G. V. (2008). Bimgaining.

HOPPE, H. (1996). Progressive meshes. *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 99–108.

HORNIK, K., STINCHCOMBE, M., & WHITE, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, *2*(5), 359–366.

HUBER, P. J. (1992). Robust estimation of a location parameter. *Breakthroughs in statistics* (pp. 492–518). Springer.

INDYK, P., & MOTWANI, R. (1998). Approximate nearest neighbors: Towards removing the curse of dimensionality. *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, 604–613.

JULIER, S. J., & UHLMANN, J. K. (1997). New extension of the kalman filter to nonlinear systems. *Signal processing, sensor fusion, and target recognition VI*, *3068*, 182–193.

KAHAN, W. (1996). Ieee standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE*, *754*(94720-1776), 11.

KARIMI, S., BRAGA, R. G., IORDANOVA, I., & ST-ONGE, D. (2021). Semantic navigation using building information on construction sites. *arXiv preprint arXiv:2104.10296*.

KARSOLIYA, S. (2012). Approximating number of hidden layer neurons in multiple hidden layer bpnn architecture. *International Journal of Engineering Trends and Technology*, *3*(6), 714–717.

KOENIG, N., & HOWARD, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, *3*, 2149–2154.

KUO, F. Y., & SLOAN, I. H. (2005). Lifting the curse of dimensionality. *Notices of the AMS*, *52*(11), 1320–1328.

KUZMIN, M. (2018). Review. classification and comparison of the existing slam methods for groups of robots. *2018 22nd Conference of Open Innovations Association (FRUCT)*, 115–120. https://doi.org/10.23919/FRUCT.2018.8468281

LATIF, Y., CADENA, C., & NEIRA, J. (2013). Robust loop closing over time for pose graph slam. *The International Journal of Robotics Research*, *32*(14), 1611–1626.

León, A. Q. (2021). *Gradient based optimization algortihms*. https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network

Lin, Y.-H., Liu, Y.-S., Gao, G., Han, X.-G., Lai, C.-Y., & Gu, M. (2013). The ifc-based path planning for 3d indoor spaces. *Advanced Engineering Informatics*, *27*(2), 189–205.

Liu, L.-M., Guo, Y.-R., Wang, Z., Yang, Z.-M., & Shao, Y.-H. (2017). K-proximal plane clustering. *International Journal of Machine Learning and Cybernetics*, *8*(5), 1537–1554.

Liu, X., He, C., Zhao, H., Jia, J., & Liu, C. (2021). Building information modeling indoor path planning: A lightweight approach for complex bim building. *Computer Animation and Virtual Worlds*, *32*(3-4), e2014.

Lopez-de-Teruel, P. E., Garcia, F. J., Canovas, O., Gonzalez, R., & Carrasco, J. A. (2017). Human behavior monitoring using a passive indoor positioning system: A case study in a sme [14th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2017) / 12th International Conference on Future Networks and Communications (FNC 2017) / Affiliated Workshops]. *Procedia Computer Science*, *110*, 182–189. https://doi.org/https://doi.org/10.1016/j.procs.2017.06.076

Low, K.-L., & Tan, T.-S. (1997). Model simplification using vertex-clustering. *Proceedings of the 1997 symposium on Interactive 3D graphics*, 75–ff.

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

Lu, F., & Milios, E. (1997). Robot pose estimation in unknown environments by matching 2d range scans. *Journal of Intelligent and Robotic systems*, *18*(3), 249–275.

MacDonald, J. D., & Booth, K. S. (1990). Heuristics for ray tracing using space subdivision. *The Visual Computer*, *6*(3), 153–166.

Meagher, D. (1982). Geometric modeling using octree encoding. *Computer graphics and image processing*, *19*(2), 129–147.

Moré, J. J. (1978). The levenberg-marquardt algorithm: Implementation and theory. *Numerical analysis* (pp. 105–116). Springer.

Moura, M. S., Rizzo, C., & Serrano, D. (2021). Bim-based localization and mapping for mobile robots in construction. *2021 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 12–18.

Muja, M., & Lowe, D. (2009). Flann-fast library for approximate nearest neighbors user manual. *Computer Science Department, University of British Columbia, Vancouver, BC, Canada*, *5*.

Naseer, T., Ruhnke, M., Stachniss, C., Spinello, L., & Burgard, W. (2015). Robust visual slam across seasons. *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2529–2535.

Oelsch, M., Karimi, M., & Steinbach, E. (2021). R-loam: Improving lidar odometry and mapping with point-to-mesh features of a known 3d reference object. *IEEE Robotics and Automation Letters*, *6*(2), 2068–2075.

Parker, L. E. (2000). Current state of the art in distributed autonomous mobile robotics. *DARS*, 3–12.

PĂTRĂUCEAN, V., ARMENI, I., NAHANGI, M., YEUNG, J., BRILAKIS, I., & HAAS, C. (2015). State of research in automatic as-built modelling. *Advanced Engineering Informatics*, *29*(2), 162–171.

Planung: Bim-modelle von jansen stahlsystemen. (2021). https://www.jansen.com/de/building-systems-stahlprofilsysteme/services-stahlsysteme/planung-bim-stahlsysteme.html

PRIETO, S., de SOTO, B. G., & ADÁN, A. (2020). A methodology to monitor construction progress using autonomous robots. *ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction*, *37*, 1515–1522.

ROUSSEEUW, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, *20*, 53–65.

RUFFLE, S. (1986). Architectural design exposed: From computer-aided drawing to computer-aided design. *Environment and Planning B: Planning and Design*, *13*(4), 385–389.

SAMET, H. (1984). The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, *16*(2), 187–260.

SHAO, Y.-H., BAI, L., WANG, Z., HUA, X.-Y., & DENG, N.-Y. (2013). Proximal plane clustering via eigenvalues. *Procedia Computer Science*, *17*, 41–47.

SHEKHAR, R., FAYYAD, E., YAGEL, R., & CORNHILL, J. F. (1996). Octree-based decimation of marching cubes surfaces. *Proceedings of Seventh Annual IEEE Visualization'96*, 335–342.

STANFORD ARTIFICIAL INTELLIGENCE LABORATORY ET AL. (2020, May 23). *Robotic operating system* (Version ROS Noetic Ninjemys). https://www.ros.org

TATENO, K., TOMBARI, F., LAINA, I., & NAVAB, N. (2017). Cnn-slam: Real-time dense monocular slam with learned depth prediction. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 6243–6252.

TAZIR, M. L., GOKHOOL, T., CHECCHIN, P., MALATERRE, L., & TRASSOUDAINE, L. (2018). Cicp: Cluster iterative closest point for sparse–dense point cloud registration. *Robotics and Autonomous Systems*, *108*, 66–86.

THRUN, S. (2002). Probabilistic robotics. *Communications of the ACM*, *45*(3), 52–57.

THRUN, S., & MONTEMERLO, M. (2006). The graph slam algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, *25*(5-6), 403–429.

UMEYAMA, S. (1991). Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, *13*(04), 376–380.

UY, M. A., & LEE, G. H. (2018). Pointnetvlad: Deep point cloud based retrieval for large-scale place recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4470–4479.

VYSOTSKA, O., & STACHNISS, C. (2016). Exploiting building information from publicly available maps in graph-based slam. *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 4511–4516.

WANG, Y., & SOLOMON, J. M. (2019). Deep closest point: Learning representations for point cloud registration. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 3523–3532.

XU, M., WEI, S., ZLATANOVA, S., & ZHANG, R. (2017). Bim-based indoor path planning considering obstacles. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, *4*.

YANG, H., SHI, J., & CARLONE, L. (2020). TEASER: Fast and Certifiable Point Cloud Registration. *IEEE Trans. Robotics*.

YANG, Z.-M., GUO, Y.-R., LI, C.-N., & SHAO, Y.-H. (2015). Local k-proximal plane clustering. *Neural Computing and Applications*, *26*(1), 199–211.

YIN, H., TANG, L., DING, X., WANG, Y., & XIONG, R. (2018). Locnet: Global localization in 3d point clouds for mobile vehicles. *2018 IEEE Intelligent Vehicles Symposium (IV)*, 728–733.

ZHANG, J., & SINGH, S. (2014). Loam: Lidar odometry and mapping in real-time. *Robotics: Science and Systems*, *2*(9).

ZHANG, J., & SINGH, S. (2015). Visual-lidar odometry and mapping: Low-drift, robust, and fast. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2174–2181.

ZHANG, W., & XIAO, C. (2019). Pcan: 3d attention map learning using contextual information for point cloud based retrieval. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 12436–12445.

Declaration

I hereby affirm that I have independently written the thesis submitted by me and have not used any sources or aids other than those indicated.

Munich, 12 Feb, 2022

Location, Date, Signature