Enhanced LiDAR Odometry and Mapping in Outdoor Environments

Zaheer Dhunny

Department of Electrical Engineering
University of Cape Town
Rondebosch, Cape Town
South Africa

Supervisor:
R.A. Verrinder

Co-Supervisor:
Dr. P. Amayo

August 2025

MSc.(Eng.) thesis submitted in partial fulfilment of the requirements for the degree of MSc. in Electrical Engineering in the Department of Electrical Engineering at the University of Cape Town

Keywords: LiDAR Odometry; 3D Maps; Motion compensation

Declaration

I, Zaheer Dhunny, hereby:

1. grant the University of Cape Town free licence to reproduce the above thesis in whole or in part, for the purpose of research;
2. declare that:
   (a) this thesis is my own unaided work, both in concept and execution, and apart from the normal guidance from my supervisors, I have received no assistance except as stated in my bibliography and acknowledgements.
   (b) neither the substance nor any part of the above thesis has been submitted in the past, or is being, or is to be submitted for a degree at this University or at any other university.

Signed by candidate

Zaheer Dhunny

Department of Electrical Engineering
University of Cape Town

Friday 1st August, 2025

Abstract

Enhanced LiDAR Odometry and Mapping in Outdoor Environments

Zaheer Dhunny

Friday 1st August, 2025

3D mapping is an essential component of autonomous navigation, yet challenges remain in scenarios involving sudden and aggressive motion. This thesis investigates the effects of abrupt changes in pitch and roll on LiDAR-based 3D maps. Experiments were conducted using a Clearpath Husky UGV at the University of Cape Town's Old Zoo, a challenging 2-hectare site featuring dense vegetation, semi-structured ruins, and unstructured terrain. The UGV was fitted with a stereo ZED 2i camera system, and a Velodyne HDL-32E. The second dataset used is the Newer College Dataset (NCD) [1] and was collected using an Ouster OS1-64 LiDAR with a 6-axis IMU. The Ouster LiDAR was held by a person walking through the campus and was abruptly rotated to cause high linear and angular accelerations. FAST-LIO2 is an accurate and real-time LiDAR inertial odometry algorithm and is the main algorithm used for this thesis. An ablation study was conducted to compare the different deskewing methods (sensor-based and modelling methods). Due to the slow speed of the Husky, the rough terrain and unstructured environment had no significant effects on the 3D map. No deskewing methods were even required which shows an accurate 3D map can be built using a relatively slow moving robot with a high frequency LiDAR. In contrast, the aggressive motions of the NCD led to mapping inaccuracies for LiDAR-only systems. This highlights the importance of motion compensation techniques involving an IMU.

Acknowledgements

I would like to thank the University of Cape Town for their financial assistance towards this research.

I would like to express my sincerest gratitude and appreciation to my supervisors, Robyn Verrinder and Dr. Paul Amayo. Thank you for your endless support, guidance and motivation for these past years.

Endless thanks to my family for all their love, support and words of encouragement from afar. Finally, my deepest and sincerest gratitude to my twin brother, Zybrann. Your patience and unconditional support has made this possible.

List of Figures

List of Tables

Chapter 1

Introduction

3D mapping remains an important topic in the field of autonomous navigation. Most mapping pipelines employ LiDAR as their primary sensor because of its ability to provide accurate and high-frequency measurements. Two key areas where LiDAR odometry is paramount are autonomous driving and search and rescue operations. In autonomous driving, LiDARs give a 3D picture of the surroundings and measure the distance to objects or pedestrians. This allows for obstacle detection (Figure 1.1). For search and rescue operations, LiDARs are used to quickly assess affected areas and identify hazards (Figure 1.2). Many LiDAR odometry methods use some form of the Iterative Closest Point (ICP) algorithm. When designing a mapping pipeline, it is essential to consider various factors, including the motion of the sensor and the environment.

Figure 1.1: The Waymo Driver includes 4 LiDARs that are used for obstacle detection [2].

Figure 1.2: DJI Matrice 350 RTK uses the Zenmuse L2 LiDAR to identify any hazard [3].

1.1 A brief background to the study

Sensor-based odometry is crucial for a robot to navigate autonomously in an unknown environment [6]. LiDARs are preferred for odometry because they are insensitive to lighting conditions [7]. Most current LiDAR-based odometry pipelines use variants of ICP to estimate the robot's pose [7]. Despite extensive research on LiDAR odometry for mapping applications, several challenges arise when designing such a pipeline. These include the motion of the robot and the characteristics of the environment.

Although LiDAR odometry pipelines offer many benefits, they face difficulties in the following situations:

- Unstructured environments: Most LiDAR odometry algorithms are based on the point-to-plane ICP algorithm, which can struggle with trees, vegetation, and other non-planar objects.
- Rapid robot motion: Quick translations or rotations of the robot can distort LiDAR points, and if these distortions are not corrected, they can lead to inaccurate pose estimation. Figure 1.3 shows the effects of motion distortion.

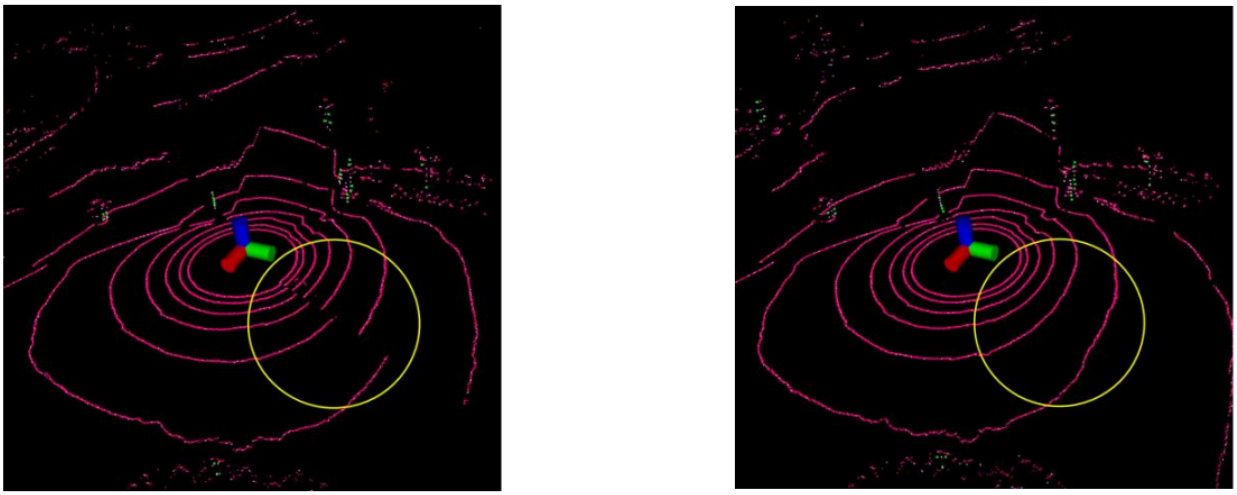

Figure 1.3: Left: scan with motion distortion; right: scan after motion compensation [4].

1.2 Problem statement

The thesis was motivated by the need for accurate 3D mapping in a challenging environment where robots experience abrupt changes in roll and pitch angles. Due to the lack of proper evaluation of LiDAR odometry pipelines, this thesis looks at implementing known algorithms and evaluating them against challenging datasets.

- Unstructured environment: Most LiDAR odometry algorithms are based on a point-to-plane ICP algorithm, and in these environments, they might suffer from the trees, vegetation and other non-planar objects.

- Quick motion of the robot: If the robot goes through a quick translation or rotation, this may cause the LiDAR points to be distorted, and if these are not corrected, will lead to increasingly inaccurate maps as the robot keeps moving.

1.3 Objective of research

The aim of the research is to investigate the impact of sudden orientation changes on the accuracy of 3D maps, assess current solutions, and propose improvements where possible. To achieve this, the research objectives are as follows:

- Evaluate current state-of-the-art LiDAR odometry systems under conditions of abrupt pitch and roll changes, using diverse real-world datasets.
- Analyze the effectiveness of existing motion compensation techniques in mitigating errors caused by rapid orientation changes.
- Construct and assess accuracy of 3D maps generated under these challenging scenarios.

1.4 Thesis scope and limitations

The scope of the thesis is to implement a LiDAR odometry and mapping pipeline to construct a 3D map in an outdoor environment. The system will also produce high-quality 3D maps in cases where the robot experiences large linear and angular accelerations, with a focus on the latter.

While the system is evaluated in an unstructured environment and abrupt motions of the robot, it is not robust to a highly dynamic environment, such as people or cars moving quickly towards the robot or for a moving surface such as Unmanned Surface Vessel (USV). The system has not been evaluated in these situations. Moreover, the solution does not consider gyroscope saturation induced by rapid motion.

Moreover, the system works on Velodyne LiDARs only or LiDARs with similar physical configurations. While the algorithm works for LiDARs with different scan lines, the configuration needs to be changed for each LiDAR. Similarly, the algorithm is designed for the coordinate systems of Velodyne LiDAR and MTI-30 IMU. If other coordinate systems are used, the algorithm would need to be modified. The input is also assumed to be coming from a rosbag.

1.5 Plan of development

This thesis is organised into six chapters as follows:

- Chapter 1 introduces the context, background, and motivation of this research, highlighting the specific challenges of LiDAR-based mapping under challenging conditions.
- Chapter 2 provides a theoretical background on LiDAR odometry algorithms, discussing foundational methods like point-to-point ICP, various ICP enhancements, and the evolution towards modern LiDAR odometry and mapping (LOAM) systems. A comparison of point-based versus feature-based matching techniques is also included.

- Chapter 3 reviews recent research on 3D mapping, focusing on advances in outdoor odometry pipelines and real-time mapping solutions.
- Chapter 4 details the specific design and implementation of the LOAM pipeline used in this thesis, describing key modules, algorithmic choices, and system configurations.
- Chapter 5 presents experimental results, demonstrating the performance and limitations of the system in real-world environments with varying degrees of motion dynamics.
- Chapter 6 concludes the thesis with a summary of findings, insights gained, and contributions.
- Chapter 7 offers recommendations for future research.

Chapter 2

Registration Algorithms Theory

In this chapter, fundamental LiDAR registration algorithms are reviewed as most state-of-the-art algorithms are built upon them. To obtain a 3D map of the environment, one must determine how much the robot moves between consecutive input data, regardless of the sensor used. The process of estimating how much the robot moves is called pose estimation or odometry. In the context fo 3D mapping, pose refers to the translation and orientation of an object relative to a fixed reference frame. A 4x4 transformation matrix is often used to describe the pose. Odometry performed using LiDAR sensors is called LiDAR odometry. Another term that is often used is LiDAR registration which refers to the process of aligning multiple LiDAR point clouds into a common coordinate system.

The most commonly used algorithms for LiDAR odometry are the Iterative Closest Point (ICP) and its variants. ICP processes two LiDAR point clouds and calculates the transformation between those two point clouds. The steps for the ICP algorithm are the following:

1. One point cloud is called the source point cloud and the other point cloud is called the target point cloud. The source point cloud will be aligned to the target point cloud which will be the reference. Another input to the ICP algorithm is the initial transformation which is the starting point for the iterative alignment process.

2. The second step involves finding the closest point in the target point cloud for each point in the reference cloud. This step is called correspondence estimation and as the name suggests, it is used to find correspondences between the two point clouds. Correspondences are pairs of points from two point clouds that are believed to represent the same points in the environment, and they serve as the basis for estimating the transformation between the two point clouds.

3. The correspondences identified in the previous step are used to calculate the transformation that aligns the source with the target. This can be done by minimising the root-mean-square (RMS) point-to-point distance between the correspondences. There are other methods used to find the transformation.

4. The estimated transformation from Step 3 is applied to the source point cloud. Doing so brings the source point cloud closer to alignment with the target point cloud.

5. This step evaluates whether the algorithm has converged which depends on the convergence criteria specified. Examples of convergence criteria are RMS point-to-point distance (between correspondences) of 0.01 m and maximum number of iterations of 100. If the

algorithm has not converged, steps 2-5 are repeated. Each iteration brings the source point cloud closer to the target point cloud until convergence is achieved.

An example of ICP registration between two 3D scans can be found in Figure 2.1 below.

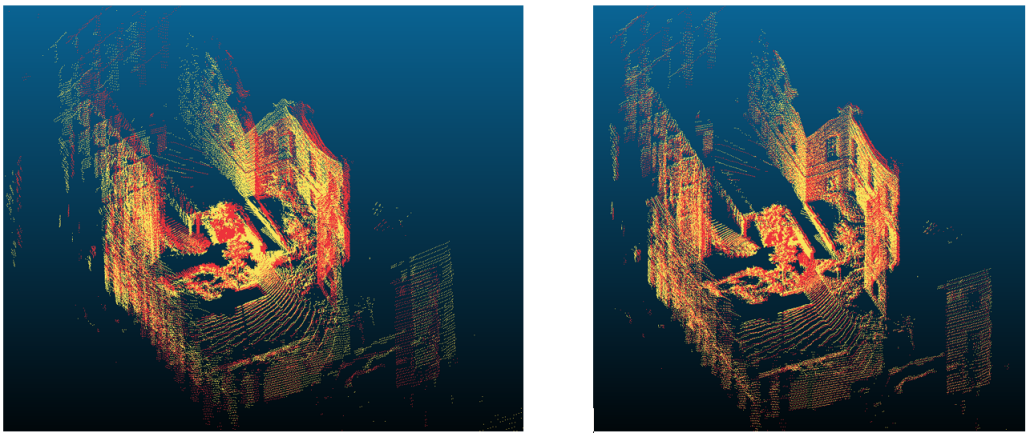

Figure 2.1: Left: alignment before ICP; right: final alignment after ICP. (Performed using CloudCompare.)

2.1 Iterative Closest Point (ICP)

ICP was first introduced in [8] and [9] to estimate the transformation between two scans. ICP relies on two iterative steps:

- Finding correspondences between the two scans
- Calculating transformation that minimises the distance between corresponding points (often referred to as the objective function) [10]

The desired transformation is often obtained after a few iterations. The standard ICP algorithm is shown in Algorithm 1 (adapted from [10]). The function FindClosestPointInA is used to find m_i, which is the matching correspondence of the transformed point using the transformation, T, in scan B and a point in A. N is the number of points in scan B and b_i refers to a point in scan B.

Algorithm 1 Point-to-Point ICP Algorithm

Require: Two scans A and B, and an initial transformation T_0
Ensure: Correct transformation T aligning A and B

$T \leftarrow T_0$
while not converged **do**
  for $i \leftarrow 1$ to N **do**
    $m_i \leftarrow \text{FindClosestPointInA}(T \cdot b_i)$
  end for
  $T \leftarrow \underset{T}{\text{argmin}} \left\{ \sum_i \|T \cdot b_i - m_i\|^2 \right\}$
end while
return T

where T is the desired transform, b_i is a point in scan B, m_i is a point in scan A that is closest to the point B transformed by T.

Chen and Medioni [9] introduced a point-to-plane ICP algorithm. The point-to-plane ICP has a faster convergence speed than the standard point-to-point ICP by taking advantage of the surface normal information of each point. The point-to-plane ICP algorithm is shown in Algorithm 2 (adapted from [10]):

Algorithm 2 Point-to-Plane ICP Algorithm

Require: Two scans A and B, and an initial transformation T_0
Ensure: Correct transformation T aligning A and B
1: $T \leftarrow T_0$
2: **while** not converged **do**
3: **for** $i \leftarrow 1$ to N **do**
4: $m_i \leftarrow \text{FindClosestPointInA}(T \cdot b_i)$
5: **end for**
6: $T \leftarrow \underset{T}{\text{argmin}} \left\{ \sum_i \|\eta_i \cdot (T \cdot b_i - m_i)\|^2 \right\}$
7: **end while**
8: **return** T

where η_i refers to the normal of m_i. Like in the previous algorithm, N is the number of points in scan B, T is the desired transform, b_i is a point in scan B, m_i is a point in scan A that is closest to the transformed point B by T.

ICP is vulnerable to outliers and [11] proposed the use of robust kernels to solve this problem. Robust functions are loss functions to reduce the effect of outliers on the desired transformation. There are many libraries such as Open3d [12] that provide modular ICP algorithms with robust kernels.

2.2 LiDAR Odometry And Mapping (LOAM)

The key concept of LOAM [13] is the use of two algorithms; odometry and mapping. The former calculates odometry at a high frequency but low accuracy to determine the LiDAR's velocity and the latter is used for fine matching and registration of point clouds at a slower rate. The odometry algorithm uses the speed of the LiDAR to remove motion distortion and the mapping algorithm matches and registers the point clouds to generate the 3D map. LOAM can be divided into three sections: scan registration, LiDAR odometry and LiDAR mapping.

2.3 Scan Registration Module

The scan registration module consists of filtering, extracting features, and adding further information to the LiDAR points, which will be useful for other modules. The 3D point clouds go through two filters: a filter to remove invalid points, which removes points with NaN coordinate values, and a proximity filter, which removes points closer than 0.1 m from the LiDAR to reduce self-scanning. The second process involves finding out which scan line each point belongs to. This is done by finding the angle that the point makes with the xy plane and based on the LiDAR, the scan line is calculated [7].

The points are further processed by finding their time in the point cloud relative to the starting point. The relative time is calculated using $\frac{\text{orientation}_{\text{point}} - \text{orientation}_{\text{start}}}{\text{orientation}_{\text{end}} - \text{orientation}_{\text{start}}}$. The orientation refers to the angle the projected point in the xy plane makes with the positive x-axis. Figure 2.2 gives an overview of the orientation.

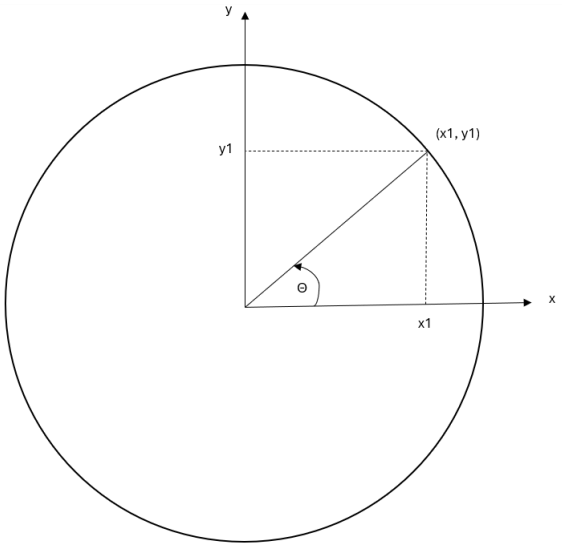

Figure 2.2: Top view of a LiDAR. The orientation angle is calculated using trigonometry.

The last process involves extracting edge points and planar points from the point cloud. The edge points, as the name suggests, refer to points lying on edges and similarly planar points refer to points lying on planes. A curvature formula as outlined in [7] is used:

$$c = \frac{1}{|S| \cdot \left\| \mathbf{X}^L_{(k,i)} \right\|} \left\| \sum_{\substack{j \in S \\ j \neq i}} \left(\mathbf{X}^L_{(k,i)} - \mathbf{X}^L_{(k,j)} \right) \right\| \tag{2.1}$$

where c refers to the curvature coefficient of the points, S refers to the points around the point, $X^L_{(k,i)}$ and $X^L_{(k,j)}$ refers to the coordinates of the point i in scan line k and point j in S in the LiDAR coordinate frame respectively. Points with the largest c values are labelled as edge points whereas points with the smallest c values are labelled as planar points. Each scan is broken down into 6 segments with each segment having a maximum 2 "good" edge points and 4 "good" planar points [7].

The edge points extraction requires sorting the points in the segment. A "good" edge point needs to be among the 2 points with the highest c values and the c value must also exceed a threshold. If a point satisfies those requirements, it is added to a "good" edge point cloud as well as the "average" edge point cloud. The latter will include the 20 points with the highest c values in each segment [7].

Similarly, the planar points extraction also requires sorting the points but from smallest to largest. A "good" planar point needs to be among the 4 points with the smallest c values and the c value must also be less than a threshold. If a point satisfies those requirements, then it is added to a "good" planar point cloud. Every point that has not been added to the edge point clouds will be added to the "average" planar point clouds and this includes the "good" planar points.

2.4 LiDAR Odometry Module

The LiDAR odometry module finds a fast estimate to the LiDAR pose between two point clouds (scan-to-scan matching). It is also used to remove distortion in the LiDAR points. The module takes in the features from the scan registration module and matches them to obtain the LiDAR pose.

The LiDAR points are first undistorted using the relative time variable from the previous module. A uniform motion model is used so that we can assume the previous pose is the same as the current pose. The transform for each point can then be calculated, $T_{point} = \frac{\text{relative time}}{\text{scan period}} * T$ where T is the transform between the latest pose and the pose at the start of the scan calculated in the LiDAR odometry module. The transform T_{point} sends all points to a common timestamp reducing motion distortion.

After deskewing, the next process is features matching and involves matching features from the current scan to the previous scan. A good edge point in the current point cloud is matched against a line comprised of normal edge points from the previous point cloud. Only two points are needed to define a line. The closest point to the good edge point is found and then the next closest point to the edge point is found but it must be close to the closest point, two nearby scans. The shortest distance or perpendicular distance between a point and a line, d_E, can be found using vector calculus, as shown in Equation (2.2) [7]:

$$d_E = \frac{\left\| \left(\tilde{\mathbf{X}}^L_{(k+1,i)} - \bar{\mathbf{X}}^L_{(k,j)} \right) \times \left(\tilde{\mathbf{X}}^L_{(k+1,i)} - \bar{\mathbf{X}}^L_{(k,l)} \right) \right\|}{\left\| \bar{\mathbf{X}}^L_{(k,j)} - \bar{\mathbf{X}}^L_{(k,l)} \right\|} \tag{2.2}$$

where:

- $\tilde{X}^L_{(k+1,i)}$ is the coordinate of the edge point i in the current scan $k+1$ in the LiDAR coordinate frame.
- $\bar{X}^L_{(k,j)}$ and $\bar{X}^L_{(k,l)}$ are the coordinates of the closest and second-closest points j and l in the previous scan k, also in the LiDAR coordinate frame.

Similarly, a good planar point in the current point cloud is matched against a plane comprised of normal planar points from the previous point cloud. We need three points to define a plane. We find the closest neighbour of i in the point cloud, denoted as j. Then, we find another two points, l and m, as the closest neighbours of i, one in the same scan of j, and the other in the two consecutive scans to the scan of j. This guarantees that the three points are non-collinear. The shortest distance or perpendicular distance between a point and a plane can be found using vector calculus, as shown in Equation (2.3) [7].

$$d_H = \frac{\left| \left(\tilde{\mathbf{X}}^L_{(k+1,i)} - \bar{\mathbf{X}}^L_{(k,j)} \right) \cdot \left[\left(\bar{\mathbf{X}}^L_{(k,j)} - \bar{\mathbf{X}}^L_{(k,l)} \right) \times \left(\bar{\mathbf{X}}^L_{(k,j)} - \bar{\mathbf{X}}^L_{(k,m)} \right) \right] \right|}{\left| \left(\bar{\mathbf{X}}^L_{(k,j)} - \bar{\mathbf{X}}^L_{(k,l)} \right) \times \left(\bar{\mathbf{X}}^L_{(k,j)} - \bar{\mathbf{X}}^L_{(k,m)} \right) \right|} \tag{2.3}$$

where:

- $\tilde{X}^L_{(k+1,i)}$ is the coordinate of the planar point i in the current scan $k+1$.

- $\bar{X}^L_{(k,j)}$, $\bar{X}^L_{(k,l)}$, and $\bar{X}^L_{(k,m)}$ are the coordinates of the three neighbouring planar points j, l, and m in the previous scan k.

2.5 LiDAR Mapping Module

The mapping module refines the pose estimated from the LiDAR odometry module and outputs the corrected pose. This module builds a 3D voxel map to store edge points and planar points. The map is represented as an array of size Width x Height x Depth where the map is divided into 21 x 21 x 11 cubes. Each cube has length of 50 m. If map has moved to the edges, it is shifted to opposite side. Feature points are added to the maps and are then downsampled. If there are more than 10 corner points and 50 surface points, then optimisation can start [14].

For each corner point, the five closest points to it in the local map are found using a k-d tree. If these five points belong on the same line, one of the three calculated eigenvalues will be significantly larger. The eigenvector corresponding to that eigenvalue is the direction of that line. Two points are obtained along this direction near the centre of the five closest points, and a residual is built using these two points and the original corner point to be matched [7].

Plane matching follows a similar approach as corner matching. Plane matching determines if 5 nearby points line on a plane using Principal Component Analysis (PCA). If these points lie on a plane, one of the eigenvalues will be relatively small, and its corresponding eigenvector is the plane normal vector. Plane fitting assumes all 5 points lie on a plane Ax + By + Cz + D = 0, where D = 1. The least squares method is used to solve the coefficients of the normal. A residual is constructed using the point to be matched, the calculated normal vector and the reciprocal of its modulus [7].

Using the newly computed pose, the feature points are transformed to the world frame. The grid position for the point is calculated and it is then inserted in the map. The 3D voxel map will be a combination of the corner map and planar map.

Chapter 3

Literature review

In this chapter, state-of-the-art LiDAR registration algorithms are reviewed with a focus on the evaluation of their robustness in the context of aggressive motions which is the main theme of this thesis. LiDAR registration algorithms can be grouped into different categories and sub-categories. The first category is LiDAR only and loosely coupled systems. Loosely coupled methods use LiDAR and IMU data separately. In the first category, there are two sub-categories: point-based and features-based methods. The second category includes algorithms that are tightly coupled. This category can be further divided into graph-based optimisation methods and Kalman filters. The taxonomy for LiDAR registration algorithms is described below:

- LiDAR-only and Loosely Coupled
  - Point-based
  - Feature-based
- Tightly coupled LiDAR-Inertial
  - Graph Optimization-based
  - Filter-based

3.1 LiDAR only and Loosely coupled LiDAR inertial Algorithms

3.1.1 Point-based methods

The point-to-point ICP [8] and the point-to-plane ICP [9] are pioneering registration algorithms and have been examined in the previous chapter. Both ICP variants stood the test of time as they are still being used after more than three decades. However, both ICP methods struggle to run in real-time when dealing with dense point clouds. Another drawback is that their sensitivity to the initial transformation. To mitigate these issues, S. Rusinkiewicz and M. Levoy [15] review the new variants since the introduction of the first and original ICP and create a new variant that runs much faster while maintaining accuracy. The authors define six stages of ICP and its variants:

- Sampling.

- Correspondence Matching
- Weighting the corresponding points
- Rejecting correspondence pairs
- Error Metric
- Error Minimisation

Rusinkiewicz and Levoy [15] compare variants of each stage based on computing time and accuracy. Various conclusions were drawn for each variant. For instance, from the experiments, it was found that point-to-plane outperforms point-to-point ICP. Rusinkiewicz and Levoy [15] also introduced a new ICP variant that can register two meshes within milliseconds.

Generalized ICP, also known as GICP, [10] is a combination of point-to-point ICP [8] and point-to-plane ICP [9] using a probabilistic model. In contrast to the point-to-point ICP and point-to-plane ICP, GICP takes into account the surface structure of both scans during registration. Particularly, GICP [10] assumes the surfaces the points lie on follow a Gaussian distribution. That is, given 2 scans, A and B, where $A = \{a_0, \ldots, a_N\}$ and $B = \{b_0, \ldots, b_N\}$, $a_i \sim \mathcal{N}(\hat{a}_i, C_{A_i})$, $b_i \sim \mathcal{N}(\hat{b}_i, C_{B_i})$. $\{C_{A_i}\}$ and $\{C_{B_i}\}$ are covariance matrices of points A and B. The objective function can be easily derived, as explained in [10]. The GICP algorithm is detailed in Algorithm 3.

Algorithm 3 GICP

Require: Two scans A and B, initial transformation T_0
Ensure: Correct transformation T aligning A and B

```
T ← T_0
while not converged do
    for i ← 1 to N do
        m_i ← FindClosestPointInA(T · b_i)
    end for
    d_i ← T · b_i − m_i
    T ← argmin_T { Σ_i d_i^T (C_i^B + T C_i^A T^T)^{-1} d_i }
end while
return T
```

where FindClosestPointinA(x) is a function that calculates the Euclidean distance between each point in scan in A and a point, x in scan B and outputs the point in scan A which corresponds to the smallest distance, N is the number of scans in each scan, d_i^T is the transformation error and is defined as $d_i^T = b_i - Ta_i$.

Experiments conducted by Segal et al. [10] found that GICP is more accurate than point-to-point ICP and is more robust to bad correspondences. Furthermore, GICP uses a plane-to-plane objective function, retaining the benefits of the point-to-plane ICP over the point-to-point ICP, such as higher accuracy.

Voxelized GICP, also known as VGICP, [16] is based on GICP [10] and is inspired by the voxelization approach of Normal Distribution Transform, also known as NDT [17]. VGICP was proposed as an improvement to GICP in order to have faster running times both with and without a GPU. While GICP uses the nearest distribution-to-distribution correspondence model, VGICP employs a voxel-based distribution-to-multi-distribution one. Using the latter

model, correspondences are found without the costly nearest neighbour search. Similar to GICP, the transform error is calculated using the correspondences however instead of having only one point correspondence, VGICP takes into account multiple correspondences. The transform error is the sum of all the transform errors of the correspondences. The algorithm finds the transformation that maximises the log-likelihood of the distribution of the sum of transform errors. The objective function is modified by replacing the mean of the distributions enabling a voxel-based approach.

VGICP was evaluated through simulated experiments and real-world experiments. The simulated experiment used the author's own simulator based on the Velodyne HDL-32e which provides both structured and unstructured environments. The methods compared were the author's GICP, PCL's GICP [18], VGICP, ICP and NDT. The first three methods completely outperformed ICP and NDT in terms of accuracy. It was found that NDT's accuracy was sensitive to the voxel resolution and provided poor accuracy when the voxel resolution was not optimal. Implementations of GICP and VGICP using multi-threading and a GPU had the lowest computation time, as expected. The real-world experiment is carried out in a parking lot using a Velodyne HDL-32e LiDAR. VGICP and the author's GICP demonstrated lowest translation and rotation errors as well as the fastest computation time. In comparison, PCL's GICP had a higher computation time due to its nearest neighbour approach.

Chen et al. [19] present Direct LiDAR Odometry (DLO) as a LiDAR registration algorithm based on GICP [10]. The 3D scans are preprocessed by applying a box filter of 1 m to remove any self-measured points and a voxel grid filter of 0.25 m in order to reduce the number of points for later scan matching. The scan matching process is divided into two parts similar to LOAM [7]. The first step involves scan-to-scan matching to obtain a rough estimate of the relative pose, and the second step involves a scan-to-map matching using the estimate from the first step to obtain the global pose. If an IMU is available, IMU pre-integration can be done to obtain an initial guess for the scan-to-scan matching. This would reduce the chance of converging to a wrong local minimum. Chen et al. [19] use a keyframe-based approach for mapping. In contrast to previous mapping strategies, DLO keeps a history of keyframes and a submap is build using a subset of those keyframes for scan-to-map matching. This differs from other radius-based methods which are computationally expensive. The keyframes consist of K-nearest neighbor (kNN) keyframe scans and nearest convex hull scans. The former gives more detail about close scans while the latter give more information about distant scans. The keyframes are not selected based on fixed translational or rotation thresholds as previous methods, but instead DLO uses an adaptive threshold for translational threshold based on a "spaciousness" metric and keeps rotation threshold fixed at 30°. This adaptive threshold works well for small environments with finer features, as it allows for more frequent capturing, and for large environments where less frequent capturing suffices due to the increased importance in the features. The registration algorithm used is NanoGICP which is inspired from FastGICP [16] and NanoFLANN [20]. DLO was evaluated against other registration methods such as BLAM [21], Cartographer [22], Lio-mapping [23], LOAM [7] and LOCUS [24] using the SubT Challenge Alpha and Beta datasets [24]. DLO had the lowest absolute pose errors out of all algorithms and lowest CPU usage, proving robustness challenging scenarios such as underground mines and abandoned subways.

S4-SLAM [25] is a LiDAR odometry system that uses three registration algorithms for water surface environments. It follows a similar two-step procedure similar to LOAM [7], where in the first step (localisation), the relative pose between two consecutive scans is found using Super4PCS and ICP. In the second step (correction), using the relative pose, the global pose between a scan and a local map is found using NDT (Normal Distributions Transform).

First of all, each point cloud is filtered to remove ground points and dynamic objects. Then, the point clouds are downsampled. To eliminate motion distortion in the point cloud, Zhou et al. [25] provide two methods: an IMU-based model and a constant velocity model. The former model computes relative pose between two scans, $\Delta s_{(k,k+1)}$ from an IMU while the latter assumes that the current pose is the same as the previous pose, $\Delta s_{(k-1,k)}$. Constant velocity model is known to be ineffective for fast changes in velocity while an IMU is the preferred method in those situations. For each point in the point cloud, linear and spherical interpolation is performed to obtain the relative transform from each point to the first point in the point cloud. Zhou et al. [12] defines the scalar c as, $c = \frac{t_i^k - t_k}{t_{scan}}$, such that $cxs_{(k,k+1)}$ is the relative transform for each point at time t_i. However, the author did not use this formula but used $c = \frac{\alpha_i^k - \alpha_k}{360}$ where α_i^k is the horizontal angle at time t_i and α_k is the horizontal angle at time t_k. The new formula for c is less accurate than the previous one as it is assumes that a scan is 360 degrees and this error will cause drifts in the final pose.

The registration process consists of two processes: localisation and correction. For localisation, an improved Super4PCS [26] algorithm is used for coarse matching. To compensate for Super4PCS's weaknesses, filtering is done to remove ground points, dynamic objects and a hands-on iterative strategy is introduced to select four coplanar points based on the evaluation of overlap rates. Then, ICP is implemented using a 3D k-d tree for fine matching. To minimise computation time for ICP, a voxel grid is used to filter the scans and the pose obtained from coarse matching is used as an initial transformation. To reduce drift that is common in scan-to-scan matching, localization matches keyframes with a dynamic voxel grid sub-map using NDT (Normal Distributions Transform). Keyframes are selected for every fifth scan and the sub-map is updated after every matching. S4-SLAM also includes loop closure and it used a k-d tree to store historical poses. If the current pose is close to a previous pose, then it becomes a candidate for loop closure. The candidate is confirmed based on the overlap rate. Super4PCS followed by ICP is used for matching and if the overlap threshold is exceeded, the candidate will be chosen. Finally the poses from odometry and loop closure are fused to obtain the final pose.

S4-SLAM is evaluated using the KITTI dataset. Motion compensation is not used here as the point clouds have already been deskewed. The results show that the improved Super4PCS algorithm outperforms its predecessor in all sequences on the KITTI dataset. The former provides reduced time cost, translation error and rotation error. While this can be satisfying, more outdoor tests are necessary as the KITTI dataset consists mainly of structured environment and displays no pitch and roll motion. S4-SLAM algorithm is evaluated in a campus environment. The algorithm uses only LiDAR point clouds and an IMU and GPS are used to obtain ground truth. The environment consists of trees, buildings, vehicles and pedestrians. [12] drifts by 0.89% without loop closure and 0.48% with loop closure. For the last experiment, the algorithm (LiDAR only) was used in a harbour environment. An Unmanned Surface Vessel (USV) equipped with a LiDAR, an IMU and a GPS was used in the experiment. [12] performed better than LOAM in all experiments carried out. This may be due to LOAM assuming a ground surface while S4-SLAM does not. S4-SLAM is able to provide LiDAR odometry with less than 1% drift with a lot of noise and on a moving platform. In all experiments carried out, it is quite disappointing that no IMU was used at all. No information was provided on the waves' data; the extent to which the USV tilted during the process. S4-SLAM represents a loosely coupled LiDAR inertial odometry system as the IMU is used only in the motion compensation stage and not used anywhere else.

IMLS-SLAM is a LiDAR registration algorithm that follows a scan-to-model matching approach. There are 3 main processes involved in IMLS-SLAM: pre-processing, sampling and scan-to-model matching. To remove motion-induced distortion in the point cloud, the relative transform

between consecutive scans is assumed to be the same. This is known as the constant velocity model and it assumes smooth angular and linear velocities of the robot. Thus, the estimated pose of the robot at time, t_k, can be found using the two previous poses, $\bar{T}(t_k) = T(t_{k-1}) * T(t_{k-2})^{-1} * T(t_{k-1})$. Interpolation is performed between the end of the previous scan, $T(t_{k-1})$ and the end of the current scan, $\bar{T}(t_k)$ to obtain the pose at time t, t ϵ $[t_{k-1}, t_k]$. A motion-compensated point cloud S_k is built using those interpolated transforms. Next, the point clouds are filtered to remove noisy ground points. This is done to remove dynamic objects such as pedestrians and vehicles using a bounding box smaller than 14 m x 14 m x 4 m. After removing these data, the ground points are added back.

The preprocessed point cloud is sampled geometrically instead of using the common random sampling method. The normals at every point are computed using the eigenvalues, λ_i obtained from PCA(Principal Component Analysis). The planar scalar, a_{2D}, is defined by: $a_{2D} = \frac{\sigma_2 - \sigma_3}{\sigma_1}$, where $\sigma_2 = \sqrt{\lambda_i}$. The following nine values are computed for each point in a scan:

$$a_{2D}^2(x_i \times \vec{n}_i) \cdot X_v \tag{3.1}$$
$$-a_{2D}^2(x_i \times \vec{n}_i) \cdot X_v \tag{3.2}$$
$$a_{2D}^2(x_i \times \vec{n}_i) \cdot Y_v \tag{3.3}$$
$$-a_{2D}^2(x_i \times \vec{n}_i) \cdot Y_v \tag{3.4}$$
$$a_{2D}^2(x_i \times \vec{n}_i) \cdot Z_v \tag{3.5}$$
$$-a_{2D}^2(x_i \times \vec{n}_i) \cdot Z_v \tag{3.6}$$
$$a_{2D}^2 \left|\vec{n}_i \cdot X_v\right| \tag{3.7}$$
$$a_{2D}^2 \left|\vec{n}_i \cdot Y_v\right| \tag{3.8}$$
$$a_{2D}^2 \left|\vec{n}_i \cdot Z_v\right| \tag{3.9}$$

where X_v, Y_v and Z_v refer to the axes of the robot frame. The first six values of point, x_i are associated with the roll,pitch and yaw of the robot while the last three values are associated with the translations of the robot. A list for each of the nine values is kept for each point in the scan, and the lists are then sorted in descending order. This results in the first point of each list having greater observability for the corresponding unknown parameters (roll, pitch, yaw and translations). During scan matching, a sample of points, x, is selected from the beginning of each list and the closest point, p_c from x, in the model is searched for. If the distance between p_c and x exceed a threshold, x is removed from the sample. This process continues until s samples are obtained. In the method, the author used s=100 points resulting in 900 points per scan.

A scan-to-model strategy is used for matching, inspired by KinectFusion [27]. But instead of using a Truncated Signed Distance Function (TSDF), an Implicit Moving Least Square (IMLS) model is built from the point cloud map of previous point clouds. The function, $I^{P_k}(x)$ is defined as the distance between a 3D point, x, to the model built from the point cloud, P_k, as shown in Equation (3.10):

$$I^{P_k}(x) = \frac{\sum_{p_i \in P_k} W_i(x) \left((x - p_i) \cdot \vec{n}_i\right)}{\sum_{p_j \in P_k} W_j(x)} \tag{3.10}$$

where p_i are points in the point cloud map, P_k, $\vec{n_i}$ are the normals at points p_i and $W_i(x)$ are the weights formulated as $W_i(x) = e^{\frac{-||x-p_i||^2}{h^2}}$ for a 3D point, x. h is a parameter used to define the

IMLS model. To match the current scan, S_k, with the point cloud map, P_k, the transform that minimises the sum of squared IMLS distances is to be determined. Because of the exponential weights, the objective function cannot be minimized. To solve this, every point in the current scan, S_k, is projected onto the IMLS model. This results in the following objective function (see Equation (3.11)):

$$\sum_{x_j \in \tilde{S}_k} |\vec{\eta_j} \cdot (Rx_j + t - y_j)|^2 \tag{3.11}$$

where $\tilde{S_k}$ is the subset of points selected from the sampling process and y_j is the projected point. The transformation is determined using the objective function define in Equation (3.11) and the scan is transformed to the global coordinate frame.

To test the lidar odometry algorithm, a Velodyne HDL-32 LiDAR is mounted on a vehicle that drives through Paris for two loops each around two km. The environment is semi-structured with roads, buildings, fences, and also vegetation. The author compared a scan-to-scan approach and IMLS-SLAM and it can be seen that the scan-to-scan approach suffers from drifts while IMLS SLAM has a drift of less than 1%. The experiment is repeated but the LiDAR is tilted by 60 degrees this time. The results were fairly comparable. The KITTI dataset was used to evaulate IMLS-SLAM further. IMLS-SLAM did not perform as well on this dataset compared to the previous one due to a more unstructured environment. IMLS-SLAM provides accurate poses when there are many planar surfaces in the environment as this is one of the assumptions made in the geometry-based sampling. IMLS-SLAM's constant velocity model has not been put to the test where the LiDAR goes through pitch and roll motion. Experiments were run to compare sampling methods and it was shown that the sampling IMLS-SLAM used is more accurate than the often used random sampling and geometric sampling methods. Further, it was demonstrated that removing dynamic objects improved performance but that improvement is not significant. In addition, larger local maps improve pose accuracy, but there is a diminishing return after 10 scans.

MC2SLAM [28] is a tightly coupled LiDAR inertial odometry system that uses an online factor graph to integrate LiDAR odometry and the IMU data. The algorithm for laser odometry uses the transformation between two consecutive scans to estimate the motion during the previous scan. For each scan, the set of laser points, S_k, go through Poisson Disk Sampling where a set of points called query points, Q_k , is selected such that no two points are closer than a minimum distance of 20 cm from each other. Random sampling is then performed to obtain N_Q points the new set, Q_k. To compensate for the motion of the LiDAR during scanning, [28] uses the query points from two consecutive scans, Q_k and Q_{k+1}. The motion compensation method calculates the relative transform of a point in its local coordinate frame to the coordinate frame of the first scan.

The initial guess for the relative transform is found by integrating the IMU. A scan-to-map approach is used instead of a scan-to-scan one due to the sparsity of the LiDAR points. According to the author, a scan-to-scan approach led to universally bad results for all datasets tested. Point-to-plane ICP is used to find the relative transform between two scans if enough neighbouring points are found within a given radius. The local map consists of the previous 100 motion-compensated scans and points are only added such that the distance between points is at least 5 cm.

The author uses the KITTI dataset to evaluate MC2SLAM. The algorithm used did not include the IMU as at the time the IMU could not be incorporated in the odometry benchmarks. The

method outperformed LOAM in all experiments tested on at the time. Moreover, the author provides his own dataset to test the algorithm in cases where pitch and roll motions are present. Four datasets are recorded and the setups for them include a head-mounted LiDAR and a car-mounted LiDAR. For the car-mounted LiDAR, two datasets are recorded, one while driving around a campus and the other while driving in a field. Two datasets are recorded for head-mounted LiDAR around a campus. The author compared the different configurations for the LiDAR system which are: scan-to-map matching with IMU integration, scan-to-scan matching with IMU integration and scan-to-map matching with no IMU integration. The methods were evaluated based on the translational drift per metre and as expected the scan-to-map matching with IMU integration outperformed all other methods in all experiments. Further, for the two car runs there is a small benefit to including the IMU while for the field trip, the largest occurs when scan-to-scan matching is used. This indicates that a scan-to-map approach may be best in outdoor environments. Moreover, for the head-mounted runs, IMU integration was essential as the drift significantly increased when no IMU was used. The author allegedly computed the translational drift per metre and rotational drift per metre for each experiment but included only the former in the paper. It would have been very useful to include the rotational drift, especially for head-mounted runs. This indicates that IMU integration should be used to manage erratic and high frequency motion. While [28] provides useful datasets, the head-mounted LiDAR experiments were conducted in a relatively structured environment (university campus). Another dataset could have been recorded to evaluate the system in an outdoor environment with fast pitch and roll motion.

Dellenbach et al. [29] present a modular LiDAR odometry pipeline, PyLiDAR-SLAM, with different LiDAR odometry paradigms, namely, classical odometry, deep LiDAR odometry and hybrid odometry. For the purpose of this thesis, only classical geometry-based registration will be considered. The author breaks down the algorithm of classical geometric LiDAR odometriy into four main fundamental steps:

- Motion initialization and preprocessing
- Map construction:
- Neighbor association
- Energy minimization

[29] uses a frame-to-model registration model using point-to-plane ICP. The author favours the point-to-plane ICP over the point-to-point ICP as the latter has a very small convergence domain. However, a good initial estimate is still needed for the algorithm to converge. As a new scan is received, it is matched to the model to find correspondences and from which, the transform is determined. The new scan is then transformed before the subsequent scan is processed.

As shown in Equation (3.12), the transform between the new scan and the reference map is found by minimising the point-to-plane distance between corresponding points.

$$E(\delta T) = \sum_i w_i \left((\delta T \cdot p_i - q_i) \cdot n_i \right) \tag{3.12}$$

where δT is the transform between the new scan and the local map, p_i are the points of the new scan, q_i and n_i are the points and normals of the local map and w_i is the weight.

The objective above closely resembles the famous point-to-plane objective with the exception of the weight, w_i. The author argues that minimising the point-to-plane objective using second order methods may be vulnerable to outliers and inaccurate correspondences. The weight, w_i can be calculated as $exp(\frac{-(p_i-q_i)^2}{\sigma^2})$ where σ is a constant set to 0.5 by the author.

The author allows the use of projective and k-d tree-based neighbour association. For the latter implementation, a k-d tree is built at every time step and correspondences and normals are found using the new scan and a local map of 30 previous point clouds . In the the projective-based implementation, the point cloud is projected onto a spherical image and the coordinates of a point are stored in each activated cell. Every new scan is projected onto a spherical image and normals are found using the neighbouring activated pixels. The local map is the same size eof the previous implementation, 30 scans, At each step, every point cloud in the map is transformed to the coordinate frame of the previous scan and then projected onto a spherical image. Correspondences are found between each point of the new scan and the vertex map (the projected point cloud onto the image) sharing same pixel values. The author prefers 3D projection to 2D as the latter suffers from problems such as occlusions and mobile objects deteriorating the map.

ICP requires a good initial prediction to converge and [29] gives two solutions for initialisation. The first implementation is the constant velocity model which uses the previous transform as an initial prediction. The other initialising strategy consists of constructing an elevation image for each new scan for point above the sensor. ORB features are then extracted and used to match consecutive scans by fitting an image homography between the two images.

PyLiDAR-SLAM is evaluated using the KITTI dataset [30], the Ford dataset [31] and the NCLT dataset [32]. The first two datasets involve driving motion and do not provide a change in roll and pitch and thus will not be considered. The classical registration algorithms (CV + Kd-F2M, EI + Kd-F2M, EI + P-F2M) are evaluated on the 01/08 NCLT sequence and the results are summarised in Table 3.1. Firstly, it was found that the CV + Kd-F2M algorithm showed the highest ATE(Average Translation Error). This demonstrates the sensitivity of the constant velocity model to abrupt changes in roll and pitch. Furthermore, it can be seen that k-d tree based search demonstrated a lower ATE (Average Translation Error) than projection-based neighbour association. This indicates that the latter is more sensitive to abrupt changes in orientation and k-d tree is the better choice under those conditions.

Method	ATE
CV + Kd-F2M	11.49
EI + Kd-F2M	1.84
EI + P-F2M	4.32

Table 3.1: ATE (Average Translational Error) from sequence 01/08 of the NCLT dataset

Continuous Time ICP or CT-ICP [33] is a LiDAR-only odometry system that follows a scan-to-map approach. It introduces a novel motion compensation method by assuming an elastic formulation of the trajectory with combined continuity in the scan matching, and discontinuity between scans. This allows the odometry system to be more robust to high frequency motion. CT-ICP is currently the best-performing LiDAR-only registration algorithm based on the KITTI dataset.

The odometry system for CT-ICP is parameterised by the pose at the beginning of the scan, Ω_b and pose at the end of the scan, Ω_e. For a time, t, during the scan, the pose of the LiDAR

is determined by interpolation and this is used to transform the point at time t to the global coordinate frame.

A grid sampling is applied on each new scan to extract keypoints that are registered in the local map. The local map consists of keypoints from previous scans stored in voxel grid. The optimal poses for two poses, Ω_b and Ω_e are obtined by solving the equation,

$$\arg \min_{X \in SE(3)^2} F_{ICP}(X) + \beta_l C_{loc}(X) + \beta_v C_{vel}(X) \tag{3.13}$$

where F_{ICP} is the scan-to-continuous-time ICP defined as:

$$F_{ICP}(X) = \frac{1}{|I^n|} \sum_i \rho\big(r_i^2[X]\big) \tag{3.14}$$

where $\rho(s)$ is a loss function to reduce the effects of outliers and r_i are the residuals of the point-to-plane distance between a point in the world coordinate frame and its closest neighbour in the local map. Specifically,

$$r_i[X] = a_i\big(p_i^W[X] - q_i^W\big) \cdot n_i \tag{3.15}$$

with

$$p_i^W[X] = R^{a_i}[X]\, p_i^L + t^{a_i}[X] \tag{3.16}$$

where the rotation and translation are interpolated as

$$R^{a_i}[X] = \text{slerp}(R_b, R_e, \alpha_i) \tag{3.17}$$

$$t^{a_i}[X] = (1 - \alpha_i)t_b + \alpha_i t_e \tag{3.18}$$

$p_i^W[X]$ is the point p_i in the world coordinate frame, n_i is the normal corresponding to $p_i^W[X]$ in the local map and p_i^L is the point p_i in the LiDAR coordinate frame. R^{a_i} and t^{a_i} are the 3D rotation and translation from the LiDAR frame to world frame at time, t_i. α_i is the interpolation factor and is defined as: $\alpha_i = \frac{t_i - t_b}{t_e - t_b}$. Spherical linear interpolation (slerp) refers to the rotational interpolation. a_i is the planar scalar of the neighbouring points of a point, p_i in a local map and is defined as $\frac{\sigma_2 - \sigma_3}{\sigma_1}$ where $\alpha_i = \sqrt{\lambda_i}$ and λ_i are the eigenvalues used to compute the normal. Compared to [34] and other methods, the weights and normals are computed from a local map instead of a point cloud.

$C_{loc}(X)$ and $C_{vel}(X)$ are the location consistency constant and the constant velocity constraint respectively. The location consistency term C_{loc} is defined in Equation (3.19) and the constant velocity constraint C_{vel}, is given in Equation (3.20)

$$C_{loc}(t_b) = \|t_b - t_e^{(n-1)}\|^2 \tag{3.19}$$

$$C_{vel}(t_b, t_e) = \|(t_e - t_b) - (t_e^{(n-1)} - t_b^{(n-1)})\|^2 \tag{3.20}$$

[33] uses a local map consisting of previous point clouds stored in voxels, arguing that it allows for faster nearest neighbourhood computation compared to k-d trees.

The loop closure implementation of CT-ICP is not analysed as the open source implementation and the experiments did not utilise it. CT-ICP employed the same loop closure method as [29].

CT-ICP is evaluated in seven different datasets, namely, KITTI, KITTI-raw, KITTI-360, KITTI-CARLA, ParisLuco, Newer College Dataset (NCD), and NCLT. The two latter datasets are the ones that will be examined as they include high frequency motions. CT-ICP is evaluated against [29] in NCD short experiment, NCD long experiment and 2012-01-08 NCLT experiment. In all experiments, CT-ICP outperformed PyLiDAR F2M as shown in t3.2. In the NCLT experiment, PyLiDAR F2M performed around 20 times worse than CT-ICP. This may have been caused by the high frequency motion of the Segway robot which presented many edge cases, where the pose at the end of the previous scan is different to the pose at the beginning of the new scan. CT-ICP accounts for these "edge" cases resulting in a low Relative Translation Error (RTE) for the NCLT experiment.

Method	01 short experiment (NCD)	02 long experiment (NCD)	2012-01-08 (NCLT)
pyLiDAR F2M	1.37	2.63	19.80
CT-ICP	0.48	0.58	1.17

Table 3.2: RTE (Relative Translational Error) from NCD and NCLT datasets

KISS-ICP [6] is a simple and effective LiDAR-only based odometry pipeline using point-to-point ICP. The point clouds from the LiDAR undergo deskewing, subsampling, correspondence estimation and registration.

To deskew the point cloud, [6] uses the constant velocity model to predict the motion of the robot. This model assumes that the linear and angular velocities of the robot in consecutive time steps are the same. Further the author assumes that the robot does not undergo significant acceleration and deceleration during the short scanning time. The author favours this model as it does not require any additional sensors compared to other methods involving wheel encoders or IMU. Moreover sensor time synchronisation and calibration are not required making for an easy set up for a LiDAR odometry system. Using the constant velocity model, the pose at time t is calculated using the previous poses at time t-1 and time t-2.

The predicted pose at time t is

$$T = \begin{pmatrix} R_{t-2}^T R_{t-1} & R_{t-2}(t_{t-1} - t_{t-2}) \\ 0 & 1 \end{pmatrix}$$

The linear and angular velocities can then be calculated as shown in Equation (3.21) and Equation (3.22) respectively.

$$v_t = \frac{R_{t-2}(t_{t-1} - t_{t-2})}{\Delta t} \tag{3.21}$$

$$\omega_t = \frac{\log\left(R_{t-2}^T R_{t-1}\right)}{\Delta t} \tag{3.22}$$

For each point, p_i, the deskewed point, p_i^*, is obtained, as shown in Equation (3.23),

$$p_i^* = \mathrm{Exp}(s_i \omega_t) p_i + s_i v_t \tag{3.23}$$

where Exp() performs slerp to calculate the rotation matrix, s_i is the relative timestamp of the point to the first point in the scans.

The author utilises the concept of double downsampling in this step. The deskewed point cloud , P^*, passes through an initial voxel filter producing the resulting point cloud, P^*_{merge}. The latter is then processed through a second voxel filter ,to obtain the double-sampled point cloud, $\hat{P}^*$. The deskewed point cloud, P^* is downsampled using a voxel grid of size αv where v is the size of the voxel grid for the local map used later in the registration step and each voxel only contains one point. αv has a maximum value of 1. The second voxel grid has size βv where $\beta \in [1.0, 2.0]$.

The deskewed and subsampled point cloud is stored in a voxel grid to represent the local map. The grid has a size v x v x v and up to N_{max} 3D points are stored in each voxel. After each registration, the voxel grid is updated by adding the newly transformed points. The voxels exceeding a range, r_{max}, are removed from the grid. To store these voxels a hash table is preferred over a 3D array for more efficient memory representation and faster nearest neighbour searches (NNS).

Traditionally, ICP methods use a maximum distance threshold in NN (nearest neigbour) data association to reject outliers. However, KISS-ICP uses an adaptive threshold. This is explained in more detail in [6]. The simple point-to-point ICP is used to register the point clouds. The author favours this variation of ICP as it does not require computations involving normals, curvatures, or other descriptors. Furthermore, these computations might be erroneous if the LiDAR points are noisy and sparse. The point cloud, $\hat{P}^*$, is transformed in the local frame using the predicted transformation from the model. The previous transform is then applied to transform the point cloud to the global frame.

KISS-ICP is evaluated against state-of-the-art LiDAR registration algorithms such as MULLS [35], F-LOAM [36] and CT-ICP [33] using the KITTI dataset [30], MulRan dataset [37], Newer College dataset [1] and NCLT dataset [32]. Only the latter two datasets are considered, as the other datasets do not provide abrupt changes in orientation. 3.3 shows the RTE of LiDAR registration algorithms for the NCLT and NCD datasets. For the NCD short and NCLT experiments, CT-ICP slightly outperforms KISS-ICP by 0.03% while for the long experiment the improvement is significant. This is due to the lack of loop closure method for KISS-ICP.

Method	NCD 01-short	NCD 02-long	NCLT 2012-01-8
MULLS	0.82	1.23	-
F-LOAM	2.02	fails	-
CT-ICP	0.48	0.58	1.17
KISS-ICP	0.51	0.96	1.27

Table 3.3: Relative translational error in % for the NCLT and NCD datasets

3.1.2 Feature-based algorithms

The key concept of LOAM [13] is the use of two algorithms; the first calculates odometry at high frequency but with low accuracy to estimate the velocity of the LiDAR and the second is used for fine matching and registration of point clouds at a slower rate. The first algorithm uses the speed of the LiDAR to remove motion distortion (as the LiDAR moves) while the mapping algorithm matches and registers the point cloud to obtain the 3D map. LOAM can be divided into 4 sections: scan registration, LiDAR odometry, LiDAR mapping and transform

integration. The LiDAR odometry process in LOAM can be divided into three parts: feature extraction, feature point correspondence and motion estimation.

Feature extraction involves classifying points returned by the laser scan as either edge points or planar points based on their curvature. This is calculated using the following formula (see Equation (3.24)):

$$c = \frac{1}{|S| \cdot \left\| X^L_{(k,i)} \right\|} \left\| \sum_{\substack{j \in S \\ j \neq i}} \left(X^L_{(k,i)} - X^L_{(k,j)} \right) \right\| \tag{3.24}$$

where i represents a point in a pointcloud, P_k, and S is the set of 3D points in the same scan.

For each scan, the points are sorted based on their smoothness or c values. The points with the largest c values are labelled as edge points and the points with the smallest c values are labelled as planar points. Furthermore, a scan is divided into four regions and each region has a maximum of two edge points and four planar points. A threshold is carefully selected so that points with smoothness values greater than the threshold is labelled as edge points, while others are classified as planar points. In addition, to filter out unreliable points, points that are located near already selected feature points are excluded. Similarly, points situated on a plane parallel to the laser beam or in occluded regions are also excluded from being selected as feature points.

The LiDAR odometry algorithm takes the previous point cloud, P_k, the current point cloud,P_{k+1}, and the pose transform from the last iteration. When a new point cloud is received, the pose transform is initially set to zero. Then, the algorithm extracts feature points from P_{k+1} to construct E_{k+1} and H_{k+1}. For each feature point, we find its correspondence in P_k. The motion estimation process is adapted to a robust fitting. The algorithm assigns a bisquare weight for each feature point. The feature points that have larger distances to their correspondences are given smaller weights while those with distances exceeding a threshold are considered as outliers and assigned with zero weights. The pose transform is then updated for one iteration. The nonlinear optimisation terminates when convergence is achieved, or the maximum number of iterations is met. If the end of the sweep is reached, the point cloud, P_{k+1}, is projected to the next timestamp, t + 2, otherwise only the current pose transform is returned.

The algorithm for mapping follows a scan-to-map matching. The mapping algorithm runs at a lower frequency than the odometry algorithm. The odometry algorithm outputs the undistorted point cloud, $\bar{P}_{k+1}$ and the LiDAR pose, T^L_{k+1}. $\bar{P}_{k+1}$ is matched and registered in the world coordinates. T^W_k is the pose of the LiDAR on the map generated by the mapping algorithm at sweep k. Using T^W_k and T^L_{k+1}, the undistorted point cloud is projected onto the map, $\bar{Q}_{k+1}$ and matched with the already existing point cloud on the map, Q_k. This is done by optimising the LiDAR pose, T^W_{k+1}.

LeGo-LOAM builds on LOAM and it is used for mapping in outdoor, unstructured environments with uneven ground. There are five stages in Lego-LOAM: segmentation, feature extraction, LiDAR odometry, LiDAR mapping and transform integration. During segmentation, the point cloud received from the LiDAR is projected onto a range image and ground points are removed from the image. Points are grouped into clusters, and each cluster is assigned a distinct label. Clusters with fewer than 30 points are ignored to remove noise (e.g., small objects such as leaves). From the segmentation algorithm, each point is assigned a label as a ground or segmented point, along with its row and column in the image and its range.

The feature extraction process is similar to [13] with the only difference being that the feature extraction in LeGo-LOAM is performed on ground points and segmented points instead of raw

3D points. A point is classified as either an edge or planar point using the same curvature formula, with 10 surrounding points considered in the calculation. A threshold value for the smoothness value is chosen as in [13] for the selection of edge and planar points.

LiDAR odometry calculates the relative pose between consecutive scans. The process is similar to [13] where edge points are matched against edges and planar points are matched against planar patches. The use of labels makes the process faster and less error-prone, as it allows verification of the label each point belongs to.

The LiDAR mapping module matches the extracted features to a local map to obtain a more accurate pose and operates at a lower frequency than LiDAR odometry similar to [13]. Pose graph optimisation is applied to LeGo-LOAM using the GTSAM library [38]. The LiDAR of each feature set is treated as a node and the feature set is considered as a measurement. Furthermore, LeGo-LOAM also includes loop closure which its predecessor does not. When a loop is detected, the current feature set is matched to the previous feature set using ICP and the pose graph is updated and sent to iSAM2 for optimisation.

F-LOAM is a LiDAR-only registration algorithm that provides a non-computationally intensive motion distortion method. It is slightly more accurate than previous LOAM variants while having less computation time. The feature extraction process is the same as that in LOAM [7]. LOAM [7] uses scan-to-scan matching for motion estimation and motion compensation which is shown to be computationally efficient due to the iterative nature of matching. F-LOAM [36] employs a two-stage method to eliminate motion distortion. The author argues that a regular LiDAR operates at a frequency of 10Hz, making it unlikely that there will be drastic changes between two consecutive scans. The constant velocity model is used and linear interpolation is applied with the previous pose to undistort all the current points. The scan-to-map matching follows is similar to LOAM [7] but F-LOAM implements a weight function using local smoothness calculated in the feature extraction process. To determine the transform, the sum of the weighted point-to-edge distance and the weighted point-to-plane distance is minimised. The global map is updated using a keyframe-based approach and comprises of the global edge and the global planar map. The second step of the motion compensation process involves using the transform from the scan-to-map matching to determine the transform between the current scan and the previous scan. Using this transform, the edge points and planar points for the current scan are undistorted and updated in the global map. The map is downsampled using a voxel grid. F-LOAM was evaluated against various state-of-the-art LiDAR registration algorithms using the KITTI dataset sequence 00-10. F-LOAM was among the algorithms with the lowest ATE and ARE and lowest computation time compared to A-LOAM, LeGO-LOAM, IMLS-SLAM, LOAM and HDL-SLAM. F-LOAM was also tested in an indoor environment and the ground truth provided by a VICON system to compute accuracy. The average localisation error was found to be less than 2 cm, which is quite accurate considering that the room is approximately 4m x 4m in size. LiDAR-based Tracking And MappINg, also known as LiTAMIN [39], is a LiDAR-only registration algorithm that uses geometry approximation in conjunction with normal distributions to reduce the computation time of ICP-based methods and increase the robustness of the loop closure. In the experiments conducted, LiTAMIN had one of the lowest computation times and the highest accuracy with loop closure.

LiTAMIN2 [40] is an improvement of LiTAMIN which significantly reduces the computation time, enabling LiDAR-based registation methods at 500-1000 Hz with the same accuracy as the best LiDAR registration methods. This is achieved by reducing the number of points used for registration while maintaining accuracy through the use of KL-divergence.

MULLS [35] is a LiDAR-only multi-metric system. MULLS can be divided into three steps: motion compensation, geometric feature points extraction and multi-metric linear least squares

ICP.

The raw 3D scans undergo a filtering process. Each point is classified as either a ground point or a non-ground point based on a dual threshold. RANSAC is used to fit the ground points onto a ground plane. A K-R neighborhood method is used to obtain a covariance matrix which is then used in the PCA analysis. Based on the eigenvectors and local linearity, planarity and curvatures, features points are classified as: facade (F), roof (R), pillar (P), beam (B), and vertex (V). A method called the Neighbourhood Category Context Coding is used to describe each vertex keypoint without any calculations. The next step is ICP registration using the target scan and reference scan which consist of features from the previous extraction process. Loop closure is implemented using [41].

While LiDAR only odometry works in most cases, it struggles in scenarios where there is a significant change in orientation. There are two types of LiDAR-interial odometry: loosely coupled and tightly coupled. In the next section, tightly coupled LiDAR registration algorithms will be reviewed. Tightly-coupled methods can be further divided into optimisation-based and filter-based approaches.

3.2 Tightly Coupled LiDAR Inertial Algorithms

LIO-Mapping [23] is a pioneering algorithm that tightly integrates LiDAR and IMU data for accurate ego-motion estimation. LIO-mapping consists of two key modules: odometry and mapping, inspired by LOAM [7].

Before each LiDAR scan, IMU state estimation and IMU pre-integration are performed. This pre-integrated IMU data is then used during the optimisation process. When a LiDAR scan is obtained, it undergoes deskewing to compensate for motion distortion followed by feature extraction to obtain edge and planar points as outlined in [7]. The local map is built using feature points with the frame of the first LiDAR scan in the local map window.

LIO-Mapping was evaluated through six experiments using a handheld Velodyne VLP-16 LiDAR mounted on an Xsens MTi-100 IMU. The conditions range from slow to fast motion, with two experiments for slow motion, two experiments for medium motion and two experiments for fast motion. Compared to state-of-the-art algorithm, LOAM, LIO-Mapping demonstrated superior accuracy, achieving lower Absolute Trajectory Error (ATE) and Absolute Rotation Error (ARE) across almost all experiments. This performance of LIO-mapping in these experiments demonstrates its robustness under rapid motion.

While LIO-Mapping provides accurate poses in most aggressive motion scenarios, it has two major drawbacks. First, the algorithm, particularly the odometry module, is computationally intensive. This results in processing time delays that prevent real-time implementation. The second drawback is its initialisation module which was taken from [42] and suffers from the same sensitivity. The initialisation process fails when rapid motion is present from the start.

In summary, LIO-mapping presents a LiDAR-inertial algorithm that is accurate in various motion scenarios. This is a significant step towards accurate pose estimation under aggressive motion. Its computationally intensive odometry module and initialisation procedure are key areas of improvement.

LIO-SAM [43] is an odometry algorithm that fuses LiDAR, IMU and optionally GPS data. The state estimation problem is defined as a Maximum A Posteriori (MAP) problem and is

represented as a factor graph. In this graph, the robot's state is represented by nodes, while the factors—IMU pre-integration, LiDAR odometry, GPS, and loop closure—serve as constraints on the nodes. A new node is added to the factor graph when pose changes by a set threshold and optimisation is then performed using iSAM2. The IMU pre-integration factor is adapted from Forster et al. [44] while the LiDAR odometry factor is taken from LeGO-LOAM [45]. To reduce drifts that occur for large maps, LIO-SAM uses GPS data. The GPS coordinates are transformed to the local coordinate frame. When a new node is added, a GPS factor is also added.

The odometry system of LIO-SAM takes in a 3D point cloud as input and performs feature extraction on it. Edge features and planar features are selected from the point cloud using a smoothness formula adapted from LOAM [7]. If the smoothness value exceeds a threshold, it is classified as an edge feature, otherwise, it is a planar feature. Calculating the smoothness value of a point requires 10 neighbouring points, with the point being at the centre of the region. The set of edge and planar features is referred to as a LiDAR frame. A LiDAR frame is added only when the pose between time steps changes by a certain threshold. The keyframe is linked to a new node, and all the LiDAR frames between two keyframes are discarded.

A sliding window approach is used to build a local map from a fixed number of recent point clouds. Instead of calculating the pose between two point clouds, this odometry method calculates the pose from a specific number of keyframes. The latter are transformed to the world frame using their associated transformations and are merged to create a voxel map. There are two voxel maps; an edge feature voxel map (voxel size = 0.2m) and a planar feature voxel map(voxel size = 0.4m). Each voxel cell can contain up to 25 points.

The next step is scan matching. The new LiDAR frame is matched against the voxel maps using LOAM to obtain the relative pose. The LiDAR frame is transformed to the world frame using the predicted motion from the IMU. Correspondences between each feature point in the LiDAR frame and the local maps are identified. To reduce long-term drift, loop closure and GPS factors are incorporated. LIO-SAM requires a 9 axis IMU and accurate extrinsic calibration to function effectively.

Direct LiDAR Inertial Odometry, also known as DLIO, [46] is tightly-coupled LiDAR-inertial algorithm that uses a geometric observer to obtain state estimates. DLIO first transforms the input data (3D scans and IMU data) to the robot's frame. The next step involves preprocessing the input scans by applying a box filter and downsampling using a voxel grid. Motion compensation is applied using the IMU data between two consecutive scans and is a two-step process from coarse to fine. IMU pre-integration is performed as outlined in [44]. The transform from the IMU pre-integration is available in discrete time, which the author defines as the coarse method. The second step involves finding a solution to the continuous-time transform. A submap is built from the keyframes of the deskewed pointclouds following [19]. GICP is then used to register the input scan with the submap. A geometric observer is subsequently used to obtain a full state estimate using the transform from the scan to map and IMU measurements.

DLIO was evaluated againt the state-of-the-art registration algorithms using the Newer College datasets [1] [47]. Four datasets were collected at the University of California but these datasets were not available at the time of writing and thus will not be considered. DLIO outperformed CT-ICP, FAST-LIO2, LIO-SAM and DLO. In addition, an ablation study was also conducted to test the effectiveness of the motion compensation method. The three methods used were: no motion compensation, discrete motion compensation using IMU pre-integration and IMU pre-integration with continuous-time motion correction. As expected, no motion compensation performed the worse out of all of them and continuous-time motion compensation performed the best.

LiDAR Inertial State Estimator, also known as LINS [48], is a lightweight odometry system that makes use of the iterated error state Extended Kalman Filter. It is one of the first tightly-coupled LiDAR-inertial odometry algorithms. The algorithm consists of three modules, namely, feature extraction, odometry and mapping. LINS uses the features extraction and mapping modules from LeGO-LOAM [45].

The core of LINS is the iterated Extended Kalman Filter. Similar to other Kalman filter variations, there is a prediction step and an update step. The current state is predicted based on IMU measurements and then corrected using LiDAR measurements. Robust state estimation is achieved by defining a local state that includes pose, velocity, biases, and local gravity. To obtain the prior state, the IMU measurements are integrated as outlined in [49].

LINS was evaluated in both indoor and outdoor experiments. For the indoor experiment, a RS-16 LiDAR was mounted on a bus and an IMU was installed inside the bus. LINS was compared to other state-of-the-art registration algorithms, namely, LeGO-LOAM [45] and LOAM [7]. The trajectory outputs from the odometry process alone and the mapping process for all three methods were also included. It was observed that LINS produced the most accurate trajectory from odometry alone. However, the mapping trajectories from all three methods were very similar. The outdoor experiments were carried out in five different environments: city, port, industrial park, forest and parking lot. LINS and LIOM (also know as LIO-Mapping) [23] were the two best-performing algorithms due to their tightly-coupled implementations. LINS achieved similar accuracy as LIOM in three of the scenarios while in the other two LINS has at least twice the accuracy as LIOM. This discrepancy may be attributed to the poor performance of LIOM in outdoor environments. The running times of LINS and LIOM were also compared, revealing that LIOM takes approximately 10 times longer to process feature points. In summary, LINS is a tightly-coupled LiDAR registration algorithm that is accurate as the state-of-the-art algorithms in both indoor and outdoor environments with significantly reduced computation time allowing real-time mapping.

FAST-LIO [50] is a tightly coupled LiDAR-inertial algorithm that uses an iterated Extended Kalman Filter to fuse LiDAR and IMU data. The author introduces a novel formula for calculating the Kalman gain to reduce computation time and a back-propagation method to undistort LiDAR points during rapid motion. Input 3D scans undergo a feature extraction process to obtain edge and planar features, similar to LOAM [7]. These features together with IMU measurements are then used as inputs in the state estimation process. The output of the state estimation module is the estimated pose derived from registering the input scans with the global map. The pose is then used to update the global map.

In order to use Kalman filters on manifold, some mathematical derivations have to be made. The full derivation can be found in [51]. The results of the derivation are two operators: $\boxplus$ and $\boxminus$ which are defined as:

$$\boxplus : M \times \mathbb{R}^n \to M \tag{3.25}$$

$$\boxminus : M \times M \to \mathbb{R}^n \tag{3.26}$$

where M is a manifold.

The continuous kinematic model of an IMU attached to a LiDAR with the IMU's frame being the reference frame, can be expressed as follows:

$$^G\dot{p}_I = {}^G v_I \tag{3.27}$$

$$ {}^{G}\dot{v}_I = {}^{G}R_I(a_m - b_a - n_a) + {}^{G}g \tag{3.28} $$

$$ {}^{G}\dot{g} = 0 \tag{3.29} $$

$$ {}^{G}\dot{R}_I = {}^{G}R_I \lfloor \omega_m - b_\omega - n_\omega \rfloor_\wedge \tag{3.30} $$

$$ \dot{b}_\omega = n_{b\omega} \tag{3.31} $$

$$ \dot{b}_a = n_{ba} \tag{3.32} $$

where ${}^{G}p_I$ and ${}^{G}R_I$ are the position and orientation of the IMU in the global frame which is the initial frame of the IMU. g is the gravity vector in the global frame, a_m and ω_m are IMU linear acceleration and angular velocity respectively, n_a and n_ω are the white noise of IMU measurements, b_a and b_ω are the IMU bias modelled as a random walk process with Gaussian noises n_{ba} and $n_{b\omega}$, and $\lfloor u \rfloor_\wedge$ is the skew-symmetric matrix that maps the cross product of u. Using the $\boxplus$ operator and the continuous kinematic model, the following discrete model can be derived:

$$ x_{i+1} = x_i \boxplus \big(\Delta t\, f(x_i, u_i, w_i)\big) \tag{3.33} $$

where i refers to the ith IMU measurement and:

$$ M = SO(3) \times \mathbb{R}^{15}, \quad \dim(M) = 18 \tag{3.34} $$

$$ x \doteq \begin{bmatrix} {}^{G}R_I^T & {}^{G}p_I^T & {}^{G}v_I^T & b_w^T & b_a^T & {}^{G}g^T \end{bmatrix}^T \tag{3.35} $$

$$ u \doteq \begin{bmatrix} w_m^T & a_m^T \end{bmatrix}^T, \quad w \doteq \begin{bmatrix} n_w^T & n_a^T & n_{bw}^T & n_{ba}^T \end{bmatrix}^T \tag{3.36} $$

$$ f(x_i, u_i, w_i) = \begin{bmatrix} w_{m_i} - b_{w_i} - n_{w_i} \\ {}^{G}v_{I_i} \\ {}^{G}R_{I_i}(a_{m_i} - b_{a_i} - n_{a_i}) + {}^{G}g_i \\ n_{bw_i} \\ n_{ba_i} \\ 0_{3\times 1} \end{bmatrix} \tag{3.37} $$

$\bar{x}_{k-1}$ is defined as the optimal state at t_{k-1} with covariance matrix, $\bar{P}_{k-1}$. $\bar{P}_{k-1}$ is the covariance of the error state vector which is defined as:

$$ \tilde{x}_{k-1} \doteq x_{k-1} \boxminus \bar{x}_{k-1} = \begin{bmatrix} \delta\theta^T & {}^{G}\tilde{p}_I^T & {}^{G}\tilde{v}_I^T & \tilde{b}_w^T & \tilde{b}_a^T & {}^{G}\tilde{g}^T \end{bmatrix}^T \tag{3.38} $$

where $\delta\theta$ is the attitude error and can be computed using $\delta\theta = Log({}^{G}\bar{R}_I^T {}^{G}R_I)$. The other errors can be easily deduced; ${}^{G}\tilde{p}_I^T = {}^{G}p_I^T - {}^{G}\bar{p}_I^T$, ${}^{G}\tilde{v}_I^T = {}^{G}v_I^T - {}^{G}\bar{v}_I^T$, $\tilde{b}_\omega^T = b_\omega^T - \bar{b}_\omega^T$,$\tilde{b}_a^T = b_a^T - \bar{b}_a^T$,${}^{G}\tilde{g}^T = {}^{G}g^T - {}^{G}\bar{g}^T$.

When IMU data is received, the propagated state is calculated by setting the noise, w_i, to zero,

$$ \hat{x}_{i+1} = \hat{x}_i \boxplus (\Delta t f(\hat{x}_i, u_i, 0)); \hat{x}_0 = \bar{x}_{k-1} \tag{3.39} $$

where $\hat{x}_k$ is defined as the propagated value of x at time t = t_k and $\Delta t = \tau_{i+1} - \tau_i$.

To obtain the propagated covariance, the error model defined below is used:

$$ \tilde{x}_{i+1} = x_{i+1} \boxminus \hat{x}_{i+1} \tag{3.40} $$

$$\tilde{x}_{i+1} = \big(x_i \boxplus \Delta t\, f(\hat{x}_i, u_i, w_i)\big) \boxminus \hat{x}_i \boxplus \big(\Delta t\, f(\hat{x}_i, u_i, 0)\big) \tag{3.41}$$

$$\tilde{x}_{i+1} \simeq F_{\tilde{x}} \tilde{x}_i + F_w w_i \tag{3.42}$$

where $F_{\tilde{x}}$ and F_w are two matrices defined in [50].

The propagated covariance is calculated as below,

$$\hat{P}_{i+1} = F_{\tilde{x}} \hat{P}_i F_{\tilde{x}}^T + F_w Q F_w^T; \hat{P}_0 = \bar{P}_{k-1} \tag{3.43}$$

where $\hat{P}_k$ represents the propagated covariance at time t = k. The forward propagation step is repeated until the end of the scan is reached.

The author uses back-propagation to compensate for motion distortion in the LiDAR points. The back-propagation process produces a relative pose, ${}^{I_k}\check{T}_{I_j}$, between the IMU frame at the time of measurement and the IMU frame at the end of the scan. Back-propagation equations are as follows:

$$\check{x}_{j-1} = \check{x}_j \boxplus \big(- \Delta t\, f(\check{x}_j, u_j, 0)\big) \tag{3.44}$$

$${}^{I_k}\check{p}_{I_{j-1}} = {}^{I_k}\check{p}_{I_j} - {}^{I_k}\check{v}_{I_j}\, \Delta t \tag{3.45}$$

$${}^{I_k}\check{v}_{I_{j-1}} = {}^{I_k}\check{v}_{I_j} - {}^{I_k}\check{R}_{I_j}\big(a_{m_{i-1}} - \check{b}_{a_k}\big)\Delta t - {}^{I_k}\check{g}_k\, \Delta t \tag{3.46}$$

$${}^{I_k}\check{R}_{I_{j-1}} = {}^{I_k}\check{R}_{I_j} Exp((\hat{b}_k - w_{m_{i-1}})\Delta t) \tag{3.47}$$

where $\check{x}_{j-1}$ is the back-propagated estimate with respect to $\check{x}_k$ and ${}^{I_k}\check{p}_{I_{j-1}}$, ${}^{I_k}\check{v}_{I_{j-1}}$ and ${}^{I_k}\check{R}_{I_{j-1}}$ refer to the propagated states. The result of the backward propagation process is the relative pose between time ρ_j and the time at the end of the scan, t_k.

$${}^{L_k}p_{f_j} = {}^{I}T_L^{-1} {}^{I_k}\check{T}_{I_j} {}^{I}T_L {}^{L_j}p_{f_j} \tag{3.48}$$

The residual is defined as the distance the estimated position of the feature point,${}^{G}\hat{p}_{f_j}^k$, to an edge or a plane, ${}^{G}q_j$, in the global map.

$$z_j^k = G_j\big({}^{G}\hat{p}_{f_j}^k - {}^{G}q_j\big) \tag{3.49}$$

where $G_j = u_j^T$ for planar features and $G_j = \lfloor u_j \rfloor_\wedge$ for edge features and u_j is the normal vector of the plane or direction vector of the edge.

$$0 = z_j^k + H_j^k \tilde{x_k^k} + v_j \tag{3.50}$$

$$J_k = \begin{bmatrix} A({}^{G}\hat{R}_{I_k}^k \boxminus {}^{G}\hat{R}_{I_k})^{-T} & 0_{3\times15} \\ I_{15\times3} & I_{15\times15} \end{bmatrix} \tag{3.51}$$

$$\hat{x_k^{k+1}} = \hat{x_k^k} \boxplus (-K z_k^k - (I - KH)(J^k)^{-1}(\hat{x_k^k} \boxminus \hat{x_k})) \tag{3.52}$$

Instead of the commonly used Kalman gain, $K = PH^T(HPH^T + R)^{-1}$, to fasten the process, a new Kalman gain is used,

$$K = (H^T R^{-1} H + P_{-1})^{-1} H^T R^{-1} \tag{3.53}$$

where P = $(J^k)^{-1}\hat{P}_k(J^k)^{-T}$

FAST-LIO has two main steps:

- Forward propagation step. Using 3.39 and 3.43, the state prediction, $\hat{x}_k$, and covariance prediction,$\hat{P}_k$, are obtained.
- Backward propagation step. Using 3.47 and 3.48, the projected point, ${}^{L_k}p_{f_j}$, is obtained.

The algorithm for FAST-LIO is detailed in Algorithm 4.

Algorithm 4 State Prediction and Update with Iterative Kalman Gain

Require: Previous state estimate $\bar{x}_{k-1}$, covariance $\bar{P}_{k-1}$, accelerometer measurement a_m, gyroscope measurement ω_m, feature point ${}^{L_j}p_{f_j}$
Ensure: Updated state estimate $\bar{x}_k$ and covariance $\bar{P}_k$
1: Using equations (3.39) and (3.43), compute predicted state $\hat{x}_k$ and covariance $\hat{P}_k$
2: Using equations (3.47) and (3.48), compute projected point ${}^{L_k}p_{f_j}$
3: Initialize $k \leftarrow -1$, and set $\hat{x}_k^{(0)} \leftarrow \hat{x}_k$
4: **repeat**
5: $k \leftarrow k + 1$
6: Compute Jacobian J_k and covariance P
7: Compute updated state estimate $\hat{x}_k^{(k+1)}$ using the new Kalman gain formula
8: **until** $\left\| \hat{x}_k^{(k+1)} \boxminus \hat{x}_k^{(k)} \right\| < \epsilon$
9: Set final state estimate $\bar{x}_k \leftarrow \hat{x}_k^{(k+1)}$
10: Set final covariance $\bar{P}_k \leftarrow (I - KH)P$
11: **return** $\bar{x}_k, \bar{P}_k$

The first experiment involved comparing the processing time between the new formula and the old formula for computing Kalman gain using the same system. From the results, it was found that the new formula reduced the computation time by a factor of 100. The second experiment evaluated the robustness of FAST-LIO using a Livox Avia LiDAR mounted on a UAV in an indoor environment. FAST-LIO was able to produce odometry outputs at a rate of 50 Hz and achieved a drift of less than 0.3% with an average running time of 6.7 ms. The third experiment assessed FAST-LIO in scenarios involving aggressive motions in an indoor environment. To simulate aggressive motions, the handheld platform was shaken rapidly. The maximum angular velocity reached was approximately 200 deg/s. FAST-LIO was compared to LOAM without an IMU and a loosely coupled LOAM. FAST-LIO produced a significantly higher quality 3D map than the LOAM methods with half the running time. This was expected as the shaking motion created distorted scans that can only be corrected using a tightly coupled system. The final experiment evaluated FAST-LIO in two outdoor environments: the main building of the University of Hong Kong and a seaport area from [48]. The drift from the University of Hong Kong experiment was less than 0.05% and for the seaport area, FAST-LIO outperformed LINS [48]. This is due to the extensive downsampling in LINS. FAST-LIO used four times more feature points than LINS.

FAST-LIO2 [5] improves FAST-LIO [50] with the addition of two new features. The first is a new data structure, that is, ikd-Tree [52]. ikd-Tree performs better overall than existing data structures such as k-d tree, Octree, nanoflann and R*-tree. The second addition is the direct registration of the raw points of each scan to the map. The implementation of FAST-LIO2 can be adapted to most sensors as feature extraction is no longer required. It also inherits the motion compensation methods and fast Kalman gain computation from FAST-LIO [50].The state formulation for FAST-LIO2 is similar to FAST-LIO but it additionally includes the online extrinsic calibration of the LiDAR and IMU.

Various experiments have been conducted to evaluate the robustness of FAST-LIO2 using different platforms such as quadrotor UAVs and handheld platforms. The two most interesting experiments are UAV flip experiment and aggressive motion using the handheld platform. The first experiment involves making the UAV flip quickly in an indoor environment. During the flip, the average angular velocity was 912 deg/s and the maximum angular velocity is 1198 deg/s. FAST-LIO2 was able to produce a very accurate 3D map despite the abrupt motion and this is largely due to its backpropagation method and its fast computation time. During the second experiment, the handheld platform was shaken back and forth to produce large orientations on a footbridge. The maximum angular velocity was 100 deg/s and the maximum velocity was 7 m/s. The drift from the experiment was less than 0.06m for a trajectory 81 m long.

FAST-LIO2 has been chosen as the preferred method due to its motion compensation method and its high accuracy in fast and aggressive motions. It can also be used with a 6-axis IMU and support various types of LiDAR sensors.

Chapter 4

Implementation

This section presents a detailed implementation of FAST-LIO2, a real-time LiDAR-Inertial odometry algorithm. FAST-LIO2 is designed to achieve accurate and robust state estimation by tightly coupling LiDAR and IMU data. The architecture of the FAST-LIO2 system consists of the following modules:

1. IMU Pre-integration
2. Scan Matching and Optimisation
3. State Estimation

4.1 Overview

FAST-LIO2 uses an iterated Extended Kalman Filter (iEKF) to create a tightly coupled LiDAR and inertial odometry system. The Extended Kalman Filter is implemented using the IKFoM toolkit [53]. This toolkit was designed for the use of iterated Extended Kalman Filter on a manifold for robotic systems. It has also been used by other odometry systems such as LINS [48], FAST-LIO [50] and Point-LIO [54].

Depending on whether feature extraction is enabled or not (FAST-LIO2 has removed the feature extraction process), point clouds will go through a feature extraction process to obtain edge and planar features, similar to LOAM variants. These features together with IMU measurements (linear accelerations and angular velocities) are then used in the state estimation process. The pose obtained from state estimation is then used to update the global map. An overview of FAST-LIO2 pipeline can be seen in Figure 4.1.

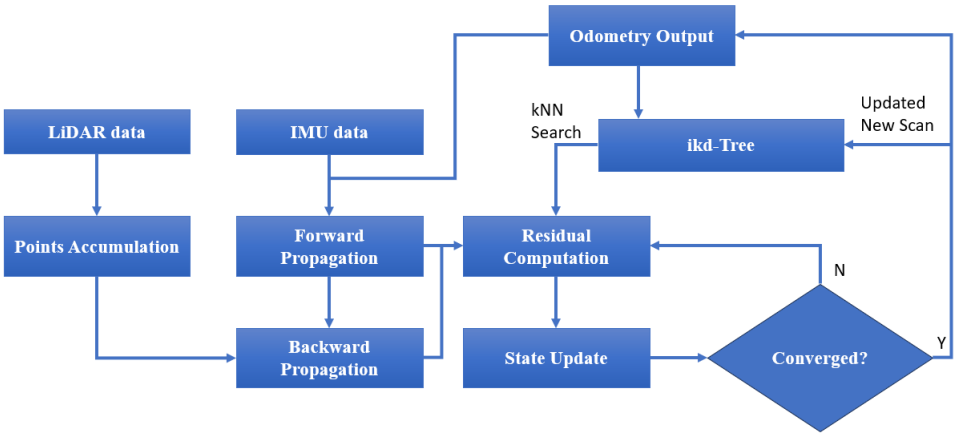

Figure 4.1: System Overview of FAST-LIO2. Adapted from [5]

4.1.1 Initialization and configuration

FAST-LIO2 requires accurate calibration between the LiDAR and IMU and it assumes the temporal offset is known beforehand. The authors of FAST-LIO [50] and FAST-LIO2 [5] recommend LI-Init [55] for calibration. LI-Init is a robust initialisation method that calibrates the offset time, extrinsic transformation between the LiDAR and IMU, the gravity vector and IMU biases. LI-Init was not used and thus depending on the experiment or dataset, some assumptions are made regarding these calibration parameters. In the next section, it is shown that these are good enough in most cases. The parameters that are required regardless of implementation are:

- time_sync_en. This parameter is used for the time synchronisation feature. When enabled, FAST-LIO2 will attempt to synchronise the timestamps of LiDAR and IMU data, either by aligning them to a common reference or by compensating for known time offsets.
- time_offset_lidar_to_imu. This parameter that specifies the time offset between the LiDAR and IMU sensors. This parameter is crucial for accurate time synchronisation and alignment of the sensor data, ensuring that the data from both sensors correspond to the same point in time.
- lidar_type. This parameter specifies the type or model of the LiDAR sensor being used. FAST-LIO2 supports Livox, Velodyne and Ouster LiDARs.
- scan_line. This refers to the number of horizontal laser lines or beams from the LiDAR.
- timestamp_unit. This specifies the unit of time used for the timestamps associated with the points in a scan.
- blind. This refers to a filter that removes points within a blind range (measured in metres) from the LiDAR.
- acc_cov. This refers to the covariance associated with the accelerometer measurements.
- gyr_cov. This refers to the covariance associated with the gyroscope measurements.
- b_acc_cov. This refers to the accelerometer bias.
- b_gyr_cov. This refers to the gyroscope bias.

- det_range. This refers to the maximum distance over which the LiDAR can effectively detect objects.
- extrinsic_est_en. This refers to the online calibration of the LiDAR and IMU. When enabled, FAST-LIO2 will determine the transformation from the LiDAR to IMU.
- extrinsic_T. This refers to the translation component of the LiDAR and IMU calibration.
- extrinsic_R. This refers to the rotation component of the calibration between the LiDAR and IMU
- point_filter_num. This is the downsampling factor.
- max_iteration. This refers to the maximum number of iterations performed by the EKF.
- filter_size_surf. This refers to the leaf size of the voxel grid for downsampling the point clouds.
- cube_side_length. This refers to the size of the 3D map.

4.2 System Modeling

4.2.1 State Transition Model

The system model consists of a state transition model and a measurement model. The state transition model predicts how the system's state evolves over time, while the measurement model links the state to the sensor observations.

G is defined as the global frame and represents the initial frame of the IMU. The IMU is referred to as the body and the latter is denoted as I while the LiDAR frame is denoted as L. The extrinsic transformation from the LiDAR to the IMU is defined by the extrinsic parameters ${}^{I}T_{L} = ({}^{I}R_{L}, {}^{I}p_{L})$, where ${}^{I}R_{L}$ represents the rotation and ${}^{I}p_{L}$ represents the translation between the two sensors. The objectives of the system are:

- To estimate the kinematic states of the system which include the position,${}^{G}p_{I}$, velocity,${}^{G}v_{I}$, and orientation,${}^{G}R_{I}$, of the IMU in the global frame.
- To estimate the other states of the system which include the IMU biases, b_a and b_ω and the gravity vector, ${}^{G}g$, in the global frame.
- To build a 3D map of the environment.

The state transition model is formulated through equations that describe the dynamics of the system, including the IMU's position, velocity, and biases [5].

$$
\begin{aligned}
{}^{G}\dot{p}_I &= {}^{G}v_I && (4.1)\\
{}^{G}\dot{v}_I &= {}^{G}R_I(a_m - b_a - n_a) + {}^{G}g && (4.2)\\
{}^{G}\dot{R}_I &= {}^{G}R_I\lfloor\omega_m - b_\omega - n_\omega\rfloor_\wedge && (4.3)\\
\dot{b}_\omega &= n_{b\omega} && (4.4)\\
\dot{b}_a &= n_{ba} && (4.5)\\
{}^{G}\dot{g} &= 0 && (4.6)\\
{}^{I}\dot{R}_L &= 0 && (4.7)\\
{}^{I}\dot{p}_L &= 0 && (4.8)
\end{aligned}
$$

where a_m, ω_m are IMU measurements and n_a, n_ω are zero mean Gaussian white noises that contribute to the IMU biases, b_a and b_ω respectively. The gravity vector,${}^{G}g$, is assumed to have a constant magnitude of 9.81 m $/s^2$.

These continuous equations are discretised at the IMU's sampling interval Δt to predict the state at the next time step:

$$
x_{i+1} = x_i \boxplus (\Delta t \cdot f(x_i, u_i, w_i)) \tag{4.9}
$$

Here, the $\boxplus$ operator denotes an operation that correctly combines state increments in the specific state space M, which includes rotations and translations. Further details on the $\boxplus$ operator can be found in [51].

The state x, input u, and noise w are defined as:

$$
\begin{aligned}
M &= SO(3) \times \mathbb{R}^{15} \times SO(3) \times \mathbb{R}^3, \dim(M) = 24\\
x &= [{}^{G}R_I^T, {}^{G}p_I^T, {}^{G}v_I^T, b_\omega^T, b_a^T, {}^{G}g^T, {}^{I}R_L^T, {}^{I}p_L^T]^T\\
u &= [\omega_m^T, a_m^T]^T\\
w &= [n_\omega^T, n_a^T, n_{b_\omega}^T, n_{b_a}^T]^T
\end{aligned} \tag{4.10}
$$

The function $f(x, u, w)$ captures the system's dynamics and is defined as:

$$
f(x,u,w) = \begin{bmatrix}
\omega_m - b_\omega - n_\omega\\
{}^{G}v_I + \frac{1}{2}({}^{G}R_I(a_m - b_a - n_a) + {}^{G}g)\Delta t\\
{}^{G}R_I(a_m - b_a - n_a) + {}^{G}g\\
n_{b_\omega}\\
n_{b_a}\\
\mathbf{0}_{3\times1}\\
\mathbf{0}_{3\times1}\\
\mathbf{0}_{3\times1}
\end{bmatrix} \tag{4.11}
$$

4.2.2 Preprocessing

The preprocessing step is quite straightforward. It takes input point clouds and first filters them by a factor of **point_filter_num**. The scans then go through another filter that removes points

that are less than **blind** m from the robot to prevent self-scanning.

4.2.3 Motion compensation

For the first ten IMU measurements, the initialisation procedure is followed. Motion compensation uses the last IMU data before the new scan and sorts the points in the LiDAR scan by time. At each IMU data, the state is forward propagated. The average acceleration as well as the covariance of the acceleration and angular velocity of the IMU data are computed, and the average acceleration is normalised to unit gravity. The Kalman filter predicts the state using the average acceleration, the angular velocity, and the covariance of white noise. Backpropagation is performed to remove motion distortion. This involves finding the points between successive IMU poses and transforming these points to the end of the LiDAR scan using only rotation. This method is not accurate as some translation is present. This process is detailed in Algorithm 5.

Algorithm 6 Process IMU Data and Undistort LiDAR Point Cloud

IMU Initialization:
if first LiDAR frame **then**
    Initialize IMU data
    Update mean accelerometer and gyroscope measurements
    Set initial IMU state and covariance
    Store last received IMU message
    Set imu_need_init_ to false if initialization is complete
end if
Undistort LiDAR Point Cloud:
Initialize variables for LiDAR undistortion
for all IMU measurements from last to first **do**
    Calculate average angular velocity and acceleration
    Normalize acceleration measurements
    Calculate time difference between current and previous IMU measurements
    Predict IMU state using Extended Kalman Filter
    Save IMU poses at each IMU measurement
    Undistort LiDAR point cloud using IMU data
end for
Update last LiDAR end time
Main Process:
Receive LiDAR and IMU data
IMU Initialization
Undistort LiDAR Point Cloud

4.2.4 Measurement Model

LiDAR devices typically collect point measurements sequentially. As a result, each point is captured at a different position if the LiDAR is in motion. To correct for motion during scanning, a technique known as backpropagation is used, as detailed in Algorithm 7. This method estimates the pose of each point in the scan relative to the end of the scan, using data from an Inertial Measurement Unit (IMU). Knowing the exact time at which each point was sampled, all points are projected as if they were collected simultaneously at the end of the scan.

Algorithm 5 UndistortPcl (LiDAR Point Cloud Undistortion)

Input: IMU measurements, LiDAR scan timestamps, initial Kalman filter state
Output: Undistorted point cloud
Initialize IMU data and append the last IMU measurement to the current IMU queue
Retrieve LiDAR scan start and end timestamps
if LiDAR type is MARSIM **then**
    Adjust LiDAR timestamps based on the previous scan
end if
Sort LiDAR points by their offset times
Forward Propagation:
Initialize IMU state and store the first pose
for each consecutive IMU measurement pair (head, tail) **do**
    if tail timestamp is before the last processed LiDAR timestamp **then**
        Continue to the next measurement pair
    end if
    Compute average angular velocity and acceleration
    Transform acceleration to account for gravity and scale
    Calculate time step (dt) based on measurement timestamps
    Update Kalman filter state with the computed values
    Store the updated pose
end for
Propagate to End Time:
Compute time step to the LiDAR scan end time
Update Kalman filter state to this timestamp
Store the final state and update timestamps
Backward Propagation for Undistortion:
for each stored IMU pose in reverse order **do**
    Extract pose data (position, velocity, acceleration, rotation)
    for each LiDAR point corresponding to the current pose **do**
        Compute undistorted position using motion compensation:
            Apply rotation and translation transformations
        Update the point's coordinates with the computed position
        if all points are processed **then**
            break
        end if
    end for
end for

k is defined as the index for LiDAR scans and $\{^{L}p_j, j = 1, \ldots, m\}$ represents the points in the k-th scan, expressed in the local LiDAR coordinate system at the scan's end time. Due to measurement noise from the LiDAR, each measured point ^{L}pj includes a noise component ^{L}nj, which arises from both range and beam direction inaccuracies. The true position of the point in the local LiDAR frame, $^{L}p_j^{gt}$, is thus:

$$^{L}p_j^{gt} = {}^{L}p_j + {}^{L}n_j \tag{4.12}$$

When projected to the global coordinate system using the LiDAR's pose $^{G}T_{I_k} = (^{G}R_{Ik}, ^{G}p_{I_k})$ and extrinsic parameters $^{I}T_L$, this true point should precisely align with a small plane segment in the map:

$$0 = {}^{G}u_j^T \left({}^{G}T_{I_k} {}^{I}T_L ({}^{L}p_j + {}^{L}n_j) - {}^{G}q_j \right), \tag{4.13}$$

where $^{G}u_j$ is the normal vector of the corresponding plane, and $^{G}q_j$ is a point on that plane. Both $^{G}T_{I_k}$ and $^{I}T_{L_k}$ are part of the state vector x_k. Therefore, the measurement from point L_{p_j} can be summarised in a more compact form:

$$0 = h_j \left(x_k, {}^{L}p_j + {}^{L}n_j \right), \tag{4.14}$$

which serves as an implicit measurement model for the state vector x_k.

As shown in Algorithm 7, h_share_model() is used to compute residuals and measurement jacobians. Each point is transformed to the world frame and for each point, the closest "surface" is found. This is done by searching for the nearest points to the target point. If there are at least five closest points and the furthest point is less than 5 m from the target point, the target point is selected for further testing. If a valid plane can be estimated from the five nearest points, the target point is selected.

Algorithm 7 h_share_model Function

```
function H_SHARE_MODEL(state_ikfom s, dyn_share_datastruct ekfom_data)
    for each point in the downsampled feature set do
        Transform the point to the world frame
        Perform nearest neighbor search
        if nearest neighbor search converges then
            Validate points based on proximity
        end if
        if no valid neighbors then
            continue to the next point
        end if
        Estimate the plane parameters and compute residual
        if plane estimation is valid then
            Calculate the score of the point
            if score exceeds threshold then
                Mark the point as selected
                Store the normal vector and residual value
            end if
        end if
    end for
    Collect effective feature points and update residuals
    for each selected point do
        Store its coordinates and associated normal vector
        Accumulate residuals
    end for
    if no effective points are found then
        Mark the data as invalid
        return
    end if
    Compute the average residual, Jacobian and measurement residual
    for each effective feature point do
        Transform the point and retrieve its normal vector
        Compute the measurement Jacobian matrix
        Calculate the distance to the estimated surface
    end for
end function
```

4.2.5 State Estimation

An iterated Extended Kalman Filter formulated on the manifold, M, is used to estimate the states defined in the state model. State estimation involves two main steps:

1. Forward propagation for each IMU measurement.
2. Iterative update for each LiDAR scan.

The optimal state estimate after the last $(k-1)_{th}$ LiDAR scan is defined as $\bar{x}_{k-1}$ and its covariance matrix is $\bar{P}_{k-1}$. When an IMU input is received, forward propagation is performed. This involves propagating the state and covariance of the system using the state model but with process noise set to zero.

$$\hat{x}_{i+1} = \hat{x}_i \boxplus (\Delta t f(\hat{x}_i, u_i, 0)); \hat{x}_0 = \bar{x}_{k-1} \tag{4.15}$$

$$\hat{P}_{i+1} = F_{\tilde{x}_i} \hat{P}_i F_{\tilde{x}_i}^T + F_{w_i} Q_i F_{w_i}^T; \hat{P}_0 = \bar{P}_{k-1} \tag{4.16}$$

where Q_i is the covariance associated with the process noise, w_i and $F_{\tilde{x}_i}$ and F_{w_i} are computed as:

$$F_{\tilde{x}_i} = \left. \frac{\partial \big(x_{i+1} \boxminus \hat{x}_{i+1}\big)}{\partial \tilde{x}_i} \right|_{\tilde{x}=0,\, w_i=0} \tag{4.17}$$

$$F_{w_i} = \left. \frac{\partial \big(x_{i+1} \boxminus \hat{x}_{i+1}\big)}{\partial w_i} \right|_{\tilde{x}=0,\, w_i=0} \tag{4.18}$$

$F_{\tilde{x}_i}$ and F_{w_i} are the Jacobians of the state model with respect to the state perturbation and the process noise, respectively. Forward propagation is repeated until the end time of the k_{th} scan, producing the propagated state $\hat{x}_k$ and covariance $\hat{P}_k$.

At the k_{th} iteration step, after motion compensation, each LiDAR point is transformed to the global frame using the following transformation:

$${}^G\hat{p}_j = {}^G\hat{T}_{I_k}^k \, {}^I\hat{T}_{L_k}^k \, {}^L p_j \tag{4.19}$$

For each transformed LiDAR point, the nearest five planar points in the ikd-Tree map are identified. These points are used to calculate the surface normal, u_i, and the centroid, ${}^Gq_i$, of the planar patch. At each iteration, the measurement model is linearised as:

$$0 = h_j(x_k, {}^L n_j) \approx h_j(\hat{x}_k^k, 0) + H_j^k \tilde{x}_k^k + v_j \tag{4.20}$$

$$= z_j^k + H_j^k \tilde{x}_k^k + v_j \tag{4.21}$$

where, $\tilde{x}_k^k = x_k \boxminus \hat{x}_k^k$, H_j^k is the Jacobian matrix of $h_j(\hat{x}_k^k \boxplus \tilde{x}_k^k, {}^L_j n_{f_j})$ with respect to $\tilde{x}_k^k$, evaluated at zero, $v_j \in \mathcal{N}(0, R_j)$ is the measurement noise, ${}^L n_j$. The residual, z_j^k, is defined as:

$$z_j^k = h_j(\hat{x}_k^k, 0) = u_j^T \big({}^G\hat{T}_{I_k}^k \, {}^I\hat{T}_{L_k}^k \, {}^L p_j - {}^G q_j\big) \tag{4.22}$$

The error state system is defined as follows:

$$x_k \boxminus \hat{x}_k = (\hat{x}_k^k \boxplus \tilde{x}_k^k) \boxminus \hat{x}_k = \hat{x}_k^k \boxminus \hat{x}_k + J_k^k \tilde{x}_k^k \sim \mathcal{N}(0, \hat{P}_k) \tag{4.23}$$

where

$$J_k = \begin{bmatrix} A(\delta^G\theta_{I_k})^{-T} & 0_{3\times15} & 0_{3\times3} & 0_{3\times3} \\ 0_{15\times3} & I_{15\times15} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times15} & A(\delta^I\theta_{L_k})^{-T} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times15} & 0_{3\times3} & I_{3\times3} \end{bmatrix} \tag{4.24}$$

Here $\delta^G\theta_{I_k} = {}^G\hat{R}^k_{I_k} \boxminus {}^G\hat{R}_{I_k}$ and $\delta^I\theta_{L_k} = {}^I\hat{R}^k_{L_k} \boxminus {}^I\hat{R}_{L_k}$. The matrix $A(u)^{-1}$, as defined in [50], is:

$$A(u)^{-1} = I - \frac{1}{2}\lfloor u \rfloor_\wedge + \left(1 - \alpha(\|u\|)\right)\frac{\lfloor u \rfloor^2_\wedge}{\|u\|^2} \tag{4.25}$$

where

$$\alpha(m) = \frac{m}{2}\cot\left(\frac{m}{2}\right) = \frac{m}{2}\frac{\cos\left(\frac{m}{2}\right)}{\sin\left(\frac{m}{2}\right)} \tag{4.26}$$

For the first iteration, $\hat{x}^k_k = \hat{x}_k$ and $J_k = I$.

Using the prior and measurement model distributions, the posteriori state x_k is computed. The maximum aposteriori estimate (MAP) is given by:

$$K = \left(H^T R^{-1} H + P^{-1}\right)^{-1} H^T R^{-1} \tag{4.27}$$

$$\hat{x}^{k+1}_k = \hat{x}^k_k \boxplus \left(-Kz^k_k - (I - KH)(J^k)^{-1}\left(\hat{x}^k_k \boxminus \hat{x}_k\right)\right) \tag{4.28}$$

$$\bar{x}_k = \hat{x}^{k+1}_k, \quad \bar{P}_k = (I - KH)P \tag{4.29}$$

After updating the state to $\bar{x}_k$, each LiDAR point ${}^Lp_j$ from the k-the scan is transformed to the global frame using the following transformation:

$${}^G\bar{p}_j = {}^G\bar{T}_{I_k}\,{}^I\bar{T}_{L_k}\,{}^Lp_j, \quad j = 1, \ldots, m. \tag{4.30}$$

Here, ${}^G\bar{T}_{I_k}$ and ${}^I\bar{T}_{L_k}$ are the transformation matrices that map the points from the local LiDAR frame to the global frame. The transformed LiDAR points $\{{}^G\bar{p}_j\}$ are then integrated into the ikd-Tree map. The algorithm for state estimation is presented below:

Algorithm 8 State Estimation

Input: Previous state estimate $\bar{x}_{k-1}$ and covariance $\bar{P}_{k-1}$;
LiDAR raw points from the current scan;
IMU data: acceleration a_m and angular velocity ω_m during the current scan.

1: Perform forward propagation to predict the current state $\hat{x}_k$ and its covariance $\hat{P}_k$.
2: Apply backward propagation to compensate for motion distortion in the LiDAR point cloud.
3: Initialize $\kappa = -1$, and set $\hat{x}_k^{(0)} = \hat{x}_k$.
4: **repeat**
5: Increment $\kappa = \kappa + 1$.
6: Calculate the Jacobian J_κ and update the covariance:

$$\hat{P}_k = (J_\kappa)^{-1} \hat{P}_k (J_\kappa)^{-T}.$$

7: Compute the residual z_κ and Jacobian H_κ based on LiDAR and IMU data.
8: Update the state estimate using:

$$\hat{x}_k^{(\kappa+1)} = \hat{x}_k^{(\kappa)} + \Delta \hat{x}_k.$$

9: **until** $\|\hat{x}_k^{(\kappa+1)} - \hat{x}_k^{(\kappa)}\| < \epsilon$
10: Set the final state estimate $\bar{x}_k = \hat{x}_k^{(\kappa+1)}$ and covariance $\bar{P}_k = (I - KH)\hat{P}_k$.
11: Transform LiDAR points to the global frame using the updated state.
Output: Optimal state estimate $\bar{x}_k$ and covariance $\bar{P}_k$;
12: Transformed LiDAR points in the global frame.

4.2.6 Map management

The data structure used to store the 3D points is the ikd-Tree [52]. It has the benefits of incremental updates such as point-wise and box-wise insertion, deletion, reinsertion, and downsampling. Map management is explained in Algorithm 10. Initially, the map is a cube with side length L m, centred around the LiDAR. When the LiDAR approaches the map boundary, the map is shifted in the direction of movement, and points outside the updated map boundary are removed from the ikd-Tree to maintain a local map.

Algorithm 9 illustrates the incremental map update process. Each point in the LiDAR frame is transformed into the global frame. Once the nearest points are identified and the EKF process is initialised, new points are added to the map. To minimise redundancy, downsampling is performed using a voxel grid-based approach. In this method, each point in the global frame is mapped to the centre of a voxel grid cell. For each new point, if the nearest existing point lies within the voxel grid cell, the new point is flagged for downsampling; otherwise, it is added to the map without downsampling. Only one representative point per voxel cell is retained to reduce data redundancy in the map.

Algorithm 9 map_incremental Function

```
1: function MAP_INCREMENTAL
2:     for each point in the downsampled point cloud do
3:         Transform the point from the body frame to the world frame
4:         if the point has nearby neighbors and system is initialized then
5:             Compute the midpoint grid location for downsampling
6:             Calculate the distance from the point to the grid midpoint
7:             if the nearest neighbor is outside the downsampling grid then
8:                 Mark the point to be added without downsampling
9:                 continue to the next point
10:            end if
11:            for each neighbor within the match threshold do
12:                if neighbor is closer to the midpoint than the current point then
13:                    Skip adding this point to the map
14:                    break the loop
15:                end if
16:            end for
17:            if the point still needs to be added then
18:                Mark the point for downsampling and addition
19:            end if
20:        else
21:            Mark the point for downsampling and addition
22:        end if
23:    end for
24:    Add points marked for downsampling to the map
25:    Add points marked without downsampling to the map
26:    Update the count of points added to the map
27:    Record the time taken for map update
28: end function
```

Algorithm 10 lasermap_fov_segment Function

```
1: function LASERMAP_FOV_SEGMENT
2:     Convert sensor position to world coordinates
3:     if the local map is not initialized then
4:         Set up map boundaries around the sensor position
5:         Mark the map as initialized
6:         return
7:     end if
8:     Check sensor position relative to map boundaries
9:     if the sensor is within boundaries then
10:        return
11:    end if
12:    Determine required shifts based on sensor position
13:    for each axis (x, y, z) do
14:        if the sensor is near a boundary then
15:            Update the map boundaries and mark regions to remove
16:        end if
17:    end for
18:    Apply the new map boundaries
19:    Remove outdated regions from the map
20:    if there are regions to remove then
21:        Update the map data structure (e.g., ikd-Tree)
22:    end if
23: end function
```

Chapter 5

Experiments

This chapter describes the experiments conducted and datasets used to evaluate the robustness of FAST-LIO2.

5.1 Introduction

State-of-the-art algorithms are evaluated using existing datasets and experiments. Different evaluation methods are used for each dataset. The robustness of FAST-LIO2 under aggressive motion conditions is evaluated in the UCT walking and rotation datasets. We also analyse the individual components of FAST-LIO2, such as the motion-compensation method and feature extraction.

Data from all the experiments were recorded in rosbag files using ROS and processed offline. The Newer College Dataset [47] is used to evaluate the system under quick rotations. The author [47] provided the ground truth transformation, T_g, and with the calculated transformation from our system, T_r, the translation error, e_t and the rotation error, e_r can be calculated as explained in [56].

To thoroughly evaluate the performance and robustness of the LiDAR-inertial system, a series of experiments using diverse datasets and experimental setups was conducted. These experiments aim to test the system under various conditions, including different environments, motion dynamics, and sensor configurations. The experiments are organised as follows:

- **HKUST Dataset**: This dataset is utilised to benchmark the system's performance in a structured indoor environment. The dataset includes ground truth poses, enabling a detailed comparison of odometry accuracy.
- **Rotation Dataset**: This dataset focuses on evaluating the system's ability to handle large rotational motions. By subjecting the system to controlled rotations, its robustness and accuracy can be assessed in scenarios with significant angular velocities.
- **Newer College Dataset (NCD)**: The NCD provides high-quality ground truth poses and maps in an outdoor environment with aggressive motion dynamics. This dataset is used to evaluate the system's performance under real-world conditions, including rapid rotations and accelerations.

- **UCT Rotation and Walking Experiments**: Conducted at the University of Cape Town, these experiments specifically target the system's response to rotational movements. By varying the speed and intensity of rotations, the impact on odometry and mapping accuracy is analysed.
- **UCT Outdoor Experiments (Zoo)**: These experiments are carried out in an outdoor environment with diverse terrain and vegetation. The aim is to test the system's robustness in unstructured environments, where factors like uneven ground and non-planar features can affect performance.

5.1.1 Accuracy Evaluation

The accuracy of each dataset is evaluated using the absolute translation error and rotation error (Root Mean Square Error (RMSE)) benchmark or the drift benchmark. If the ground truth of the dataset is given, the former criterion is used, while the drift benchmark is used where no ground truth is available. The translation error and rotation error can be obtained from Equations 5.1 and 5.2. The drift is measured between the start and end points which should be approximately zero for UCT-based experiments as they start and end at the same point.

$$\Delta T = T_r T_g^{-1} = \begin{bmatrix} \Delta R & \Delta t \\ 0 & 1 \end{bmatrix} \tag{5.1}$$

$$e_t = \left\| \Delta t \right\| = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2} \tag{5.2}$$

5.2 HKUST Dataset

The HKUST dataset, as referenced in [23], is used to evaluate the performance of FAST-LIO2 under varying motion speeds in a structured environment. The dataset is recorded using a Velodyne VLP-16 LiDAR, an Xsens MTI-100 IMU, and a motion capture system to obtain the ground truth. The dataset sequences are categorised based on the LiDAR's motion speed: slow1, slow2, med1, med2, fast1, and fast2. The sensor platform in this dataset moves abruptly, providing a challenging scenario for LiDAR odometry algorithms. Figure 5.1 illustrates the angular velocity and linear acceleration during the "fast2" experiment, where the angular velocity frequently exceeds 300 degrees per second.

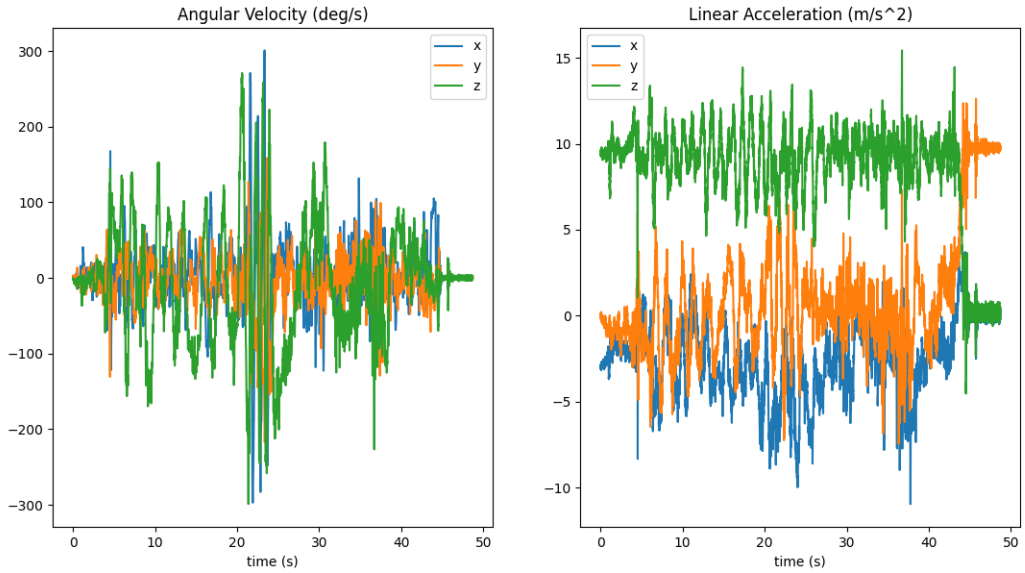

Figure 5.1: The angular velocity and acceleration in the fast2 experiment.

5.2.1 Odometry Evaluation

Both FAST-LIO2 and A-LOAM were tested across all dataset sequences to compare their performance. FAST-LIO2 relies heavily on accurate point timestamps for precise motion correction, whereas A-LOAM does not. The HKUST dataset does not include point timestamps, requiring FAST-LIO2 to estimate these timestamps, which introduces significant challenges. The parameters configured for FAST-LIO2 were as follows:

FAST-LIO2 Parameters

```
Minimum rannge distance: 1
Number of scan lines: 16
Point filter number: 4
filter size surf: 0.5
filter size map: 0.5
```

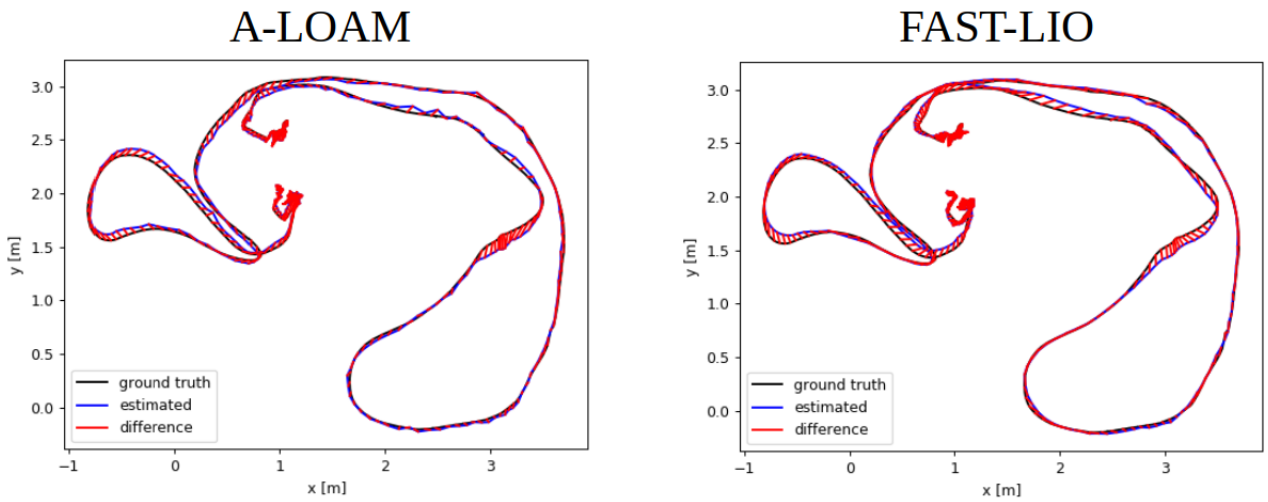

Figure 5.2: Trajectories generated by A-LOAM and FAST-LIO2 for slow1 dataset.

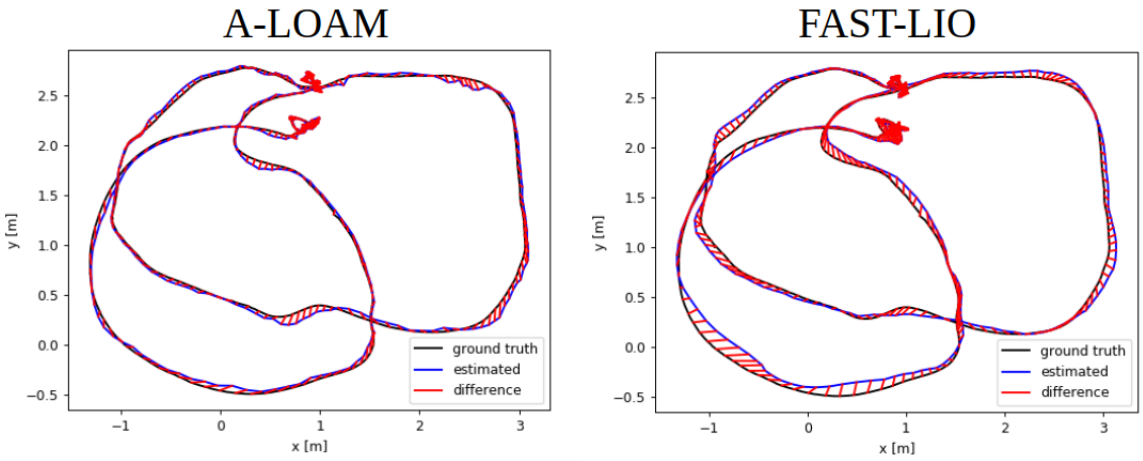

Figure 5.3: Trajectories generated by A-LOAM and FAST-LIO2 for slow2 dataset.

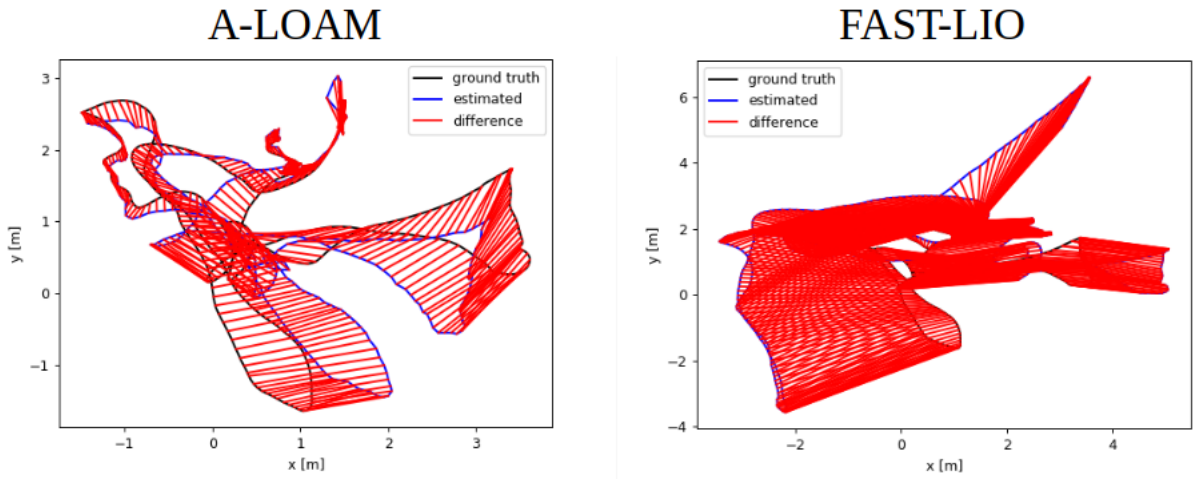

Figure 5.4: Trajectories generated by A-LOAM and FAST-LIO2 for med1 dataset.

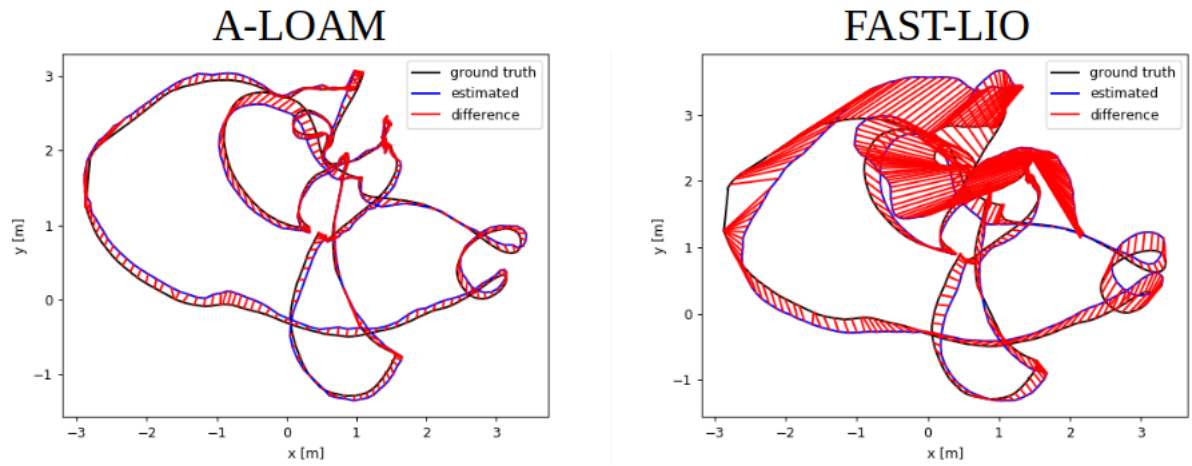

Figure 5.5: Trajectories generated by A-LOAM and FAST-LIO2 for med2 dataset.

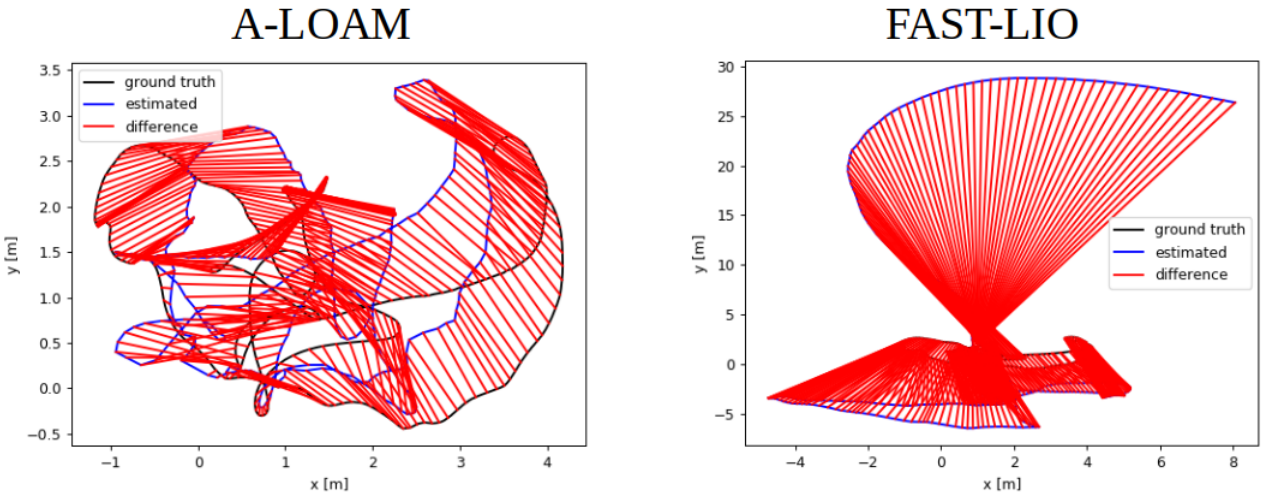

Figure 5.6: Trajectories generated by A-LOAM and FAST-LIO2 for fast1 dataset.

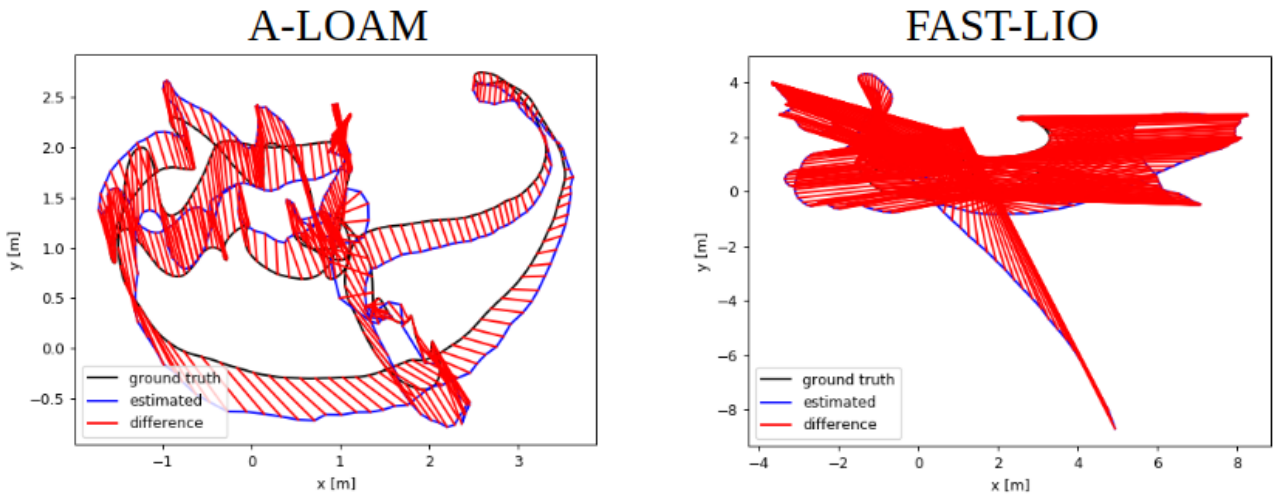

Figure 5.7: Trajectories generated by A-LOAM and FAST-LIO2 for fast2 dataset.

Both A-LOAM and FAST-LIO2 can be seen to struggle in fast environments as shown in Figures 5.6 and 5.7. Deskewing cannot be applied correctly in the case of FAST-LIO2 due to

the missing point timestamps from the dataset. As outlined in Table 5.1, A-LOAM consistently outperforms FAST-LIO2. This discrepancy is primarily due to FAST-LIO2's reliance on accurate point timestamps for deskewing and motion correction, which are not available in the HKUST dataset. Consequently, FAST-LIO2 must estimate these timestamps, leading to significant errors.

Errors	Sequence	A-LOAM	FAST-LIO2
Translation RMSE (m)	fast 1	**1.035785**	14.236272
	fast 2	**0.429002**	4.392921
	med 1	**0.488617**	2.272162
	med 2	**0.124117**	0.604300
	slow 1	**0.066218**	0.081074
	slow 2	**0.069051**	0.107827
Rotation RMSE (rad)	fast 1	**1.2215501**	13.143630
	fast 2	**0.645249**	5.949137
	med 1	**0.549929**	2.631730
	med 2	**0.282929**	0.766594
	slow 1	**0.148336**	0.183028
	slow 2	**0.146734**	0.208854

Table 5.1: Translational and rotational errors of A-LOAM and FAST-LIO2 with respect to the ground-truth from running the HKUST dataset.

Both A-LOAM and FAST-LIO2 show robust performance in slower motion scenarios, closely matching the ground truth trajectories as shown in Figures 5.2, 5.3, 5.4 and 5.5. However, as the speed increases, FAST-LIO2 struggles due to the absence of point timestamps, leading to significant errors in translation and rotation. This highlights the importance of accurate timestamping for high-speed odometry in LiDAR-inertial systems.

5.3 Rotation Dataset

The Rotation dataset, as referenced in [43], is designed to test the robustness of LiDAR odometry systems under aggressive sensor movements. This dataset was collected using a Velodyne VLP-16 LiDAR and a MicroStrain 3DM-GX5-25 IMU, with the sensor suite held by a stationary individual who rapidly rotated it in an outdoor, structured environment.

The primary challenge in this dataset is the high-speed, abrupt rotation of the sensor suite, which tests the algorithms' ability to maintain accurate odometry under rapid changes in angular velocity. KISS-ICP failed to provide accurate results, as shown in Figure 5.8. The initial point clouds include abrupt rotations, and KISS-ICP assumes no motion in the first two point clouds. This assumption leads to an inaccurate local map, causing erroneous transformations during registration with that local map. Due to these issues, the map produced by KISS-ICP is not displayed, as KISS-ICP failed to produce meaningful results.

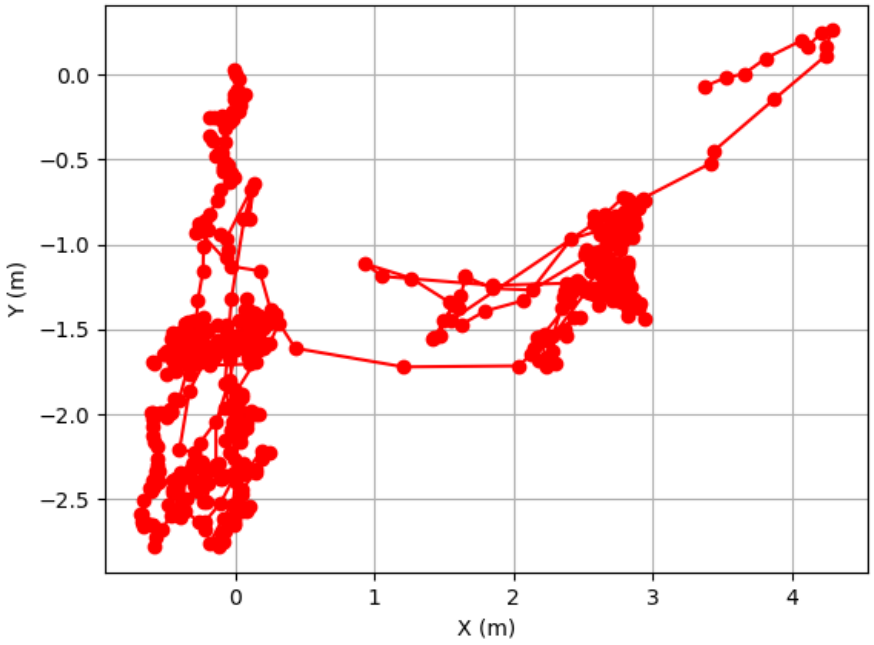

Figure 5.8: Trajectory generated by KISS-ICP on the Rotation dataset.

In contrast, FAST-LIO2 and LIO-SAM demonstrated robust performance on this dataset, effectively handling aggressive rotations due to their tightly coupled LiDAR-IMU systems. This tightly coupled approach enables both algorithms to compensate for fast motion by integrating high-frequency IMU data with LiDAR observations. This process provides continuous pose updates and is used for motion compensation.

Figure 5.9 shows FAST-LIO2's trajectory aligned closely with the LIO-SAM ground truth on the Rotation dataset. This alignment demonstrates the method's robustness under aggressive rotational motion, with minimal drift observed in high speed rotations. Figure 5.10 compares the mapping accuracy, with red regions indicating significant errors and blue regions indicating areas with low error. Point-to-point distances between maps produced by LIO-SAM and FAST-LIO2 were calculated, with lower values suggesting minimal errors and higher values indicating deviations from the ground truth. As shown in Figure 5.11, the histogram reveals predominantly low distance values, indicating high mapping accuracy. This is confirmed by the mapping accuracy visualization, where blue regions indicate high accuracy, with little green and no red present. These results demonstrate FAST-LIO2's robustness under challenging conditions.

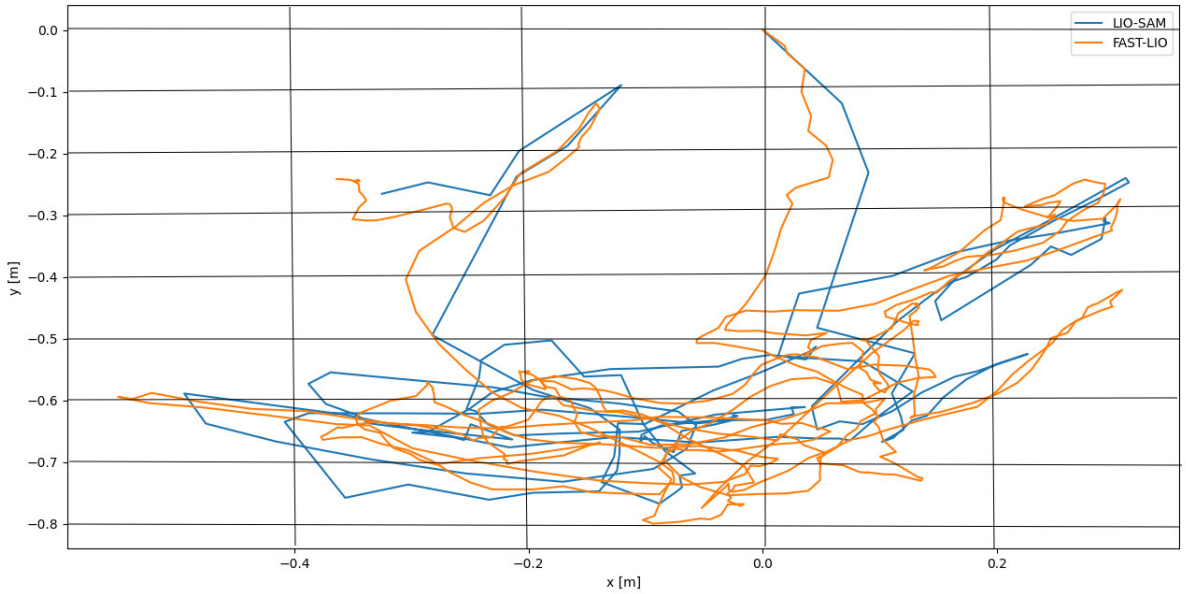

Figure 5.9: FAST-LIO2 trajectory (vs. LIO-SAM ground truth) on the Rotation dataset, demonstrating robustness under aggressive motion.

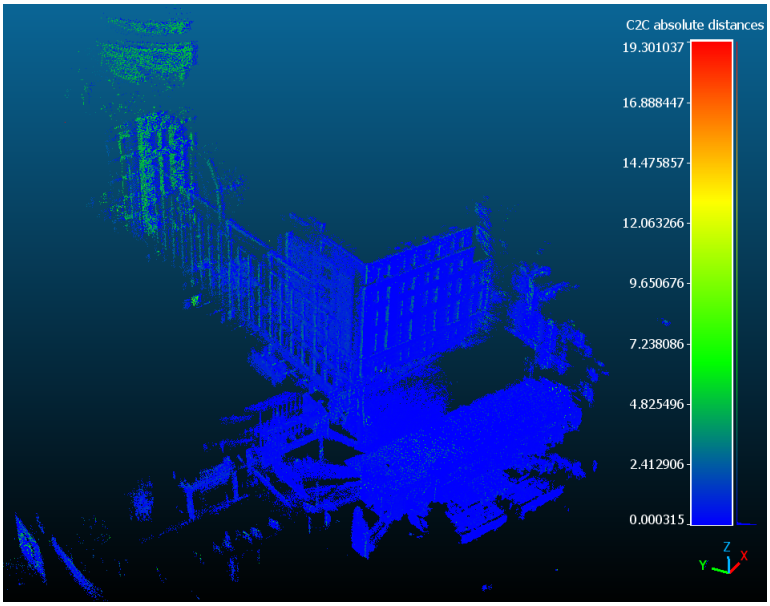

Figure 5.10: FAST-LIO2 mapping accuracy on Rotation dataset. Red: significant errors relative to LIO-SAM.

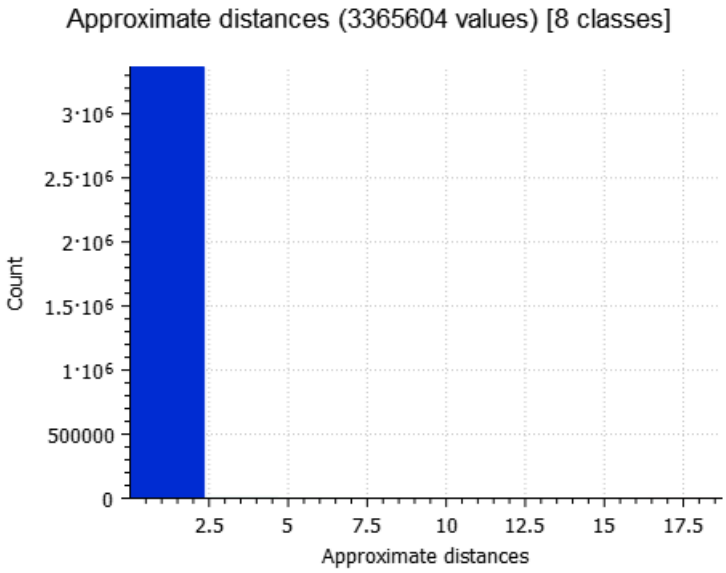

Figure 5.11: Histogram of point-to-point distances between FAST-LIO2 and LIO-SAM ground truth maps (Rotation dataset).

In conclusion, both FAST-LIO2 and LIO-SAM exhibit robust performance in handling aggressive rotations, achieving accurate odometry where KISS-ICP fails. This performance highlights the effectiveness of tightly coupled LiDAR-IMU systems, continuous pose refinement through IMU data, and motion compensation. Their successful results confirm their suitability for scenarios with rapid sensor movements, underscoring the critical role of integrated IMU data in achieving accurate odometry under challenging conditions.

5.4 Newer College Dataset

The 2021 Newer College Dataset [47] is chosen to further evaluate the LiDAR-inertial system, as it involves challenging rotations involved and it provides high-quality ground truth poses and maps. The experiments have been carried out using a custom-built platform featuring the Ouster OS0-128 LiDAR which is equipped with an internal 6-axis IMU(ICM-20948). The LiDAR outputs data every 0.1 seconds while the IMU outputs data every 0.01 seconds. The platform also incorporates a camera with an integrated IMU. The datasets from the New College and Maths institutes exhibit aggressive motion, as evidenced by the analysis of each dataset. Figures 5.12 to 5.17 show that during the experiments the angular velocities exceed 200 degrees per second.

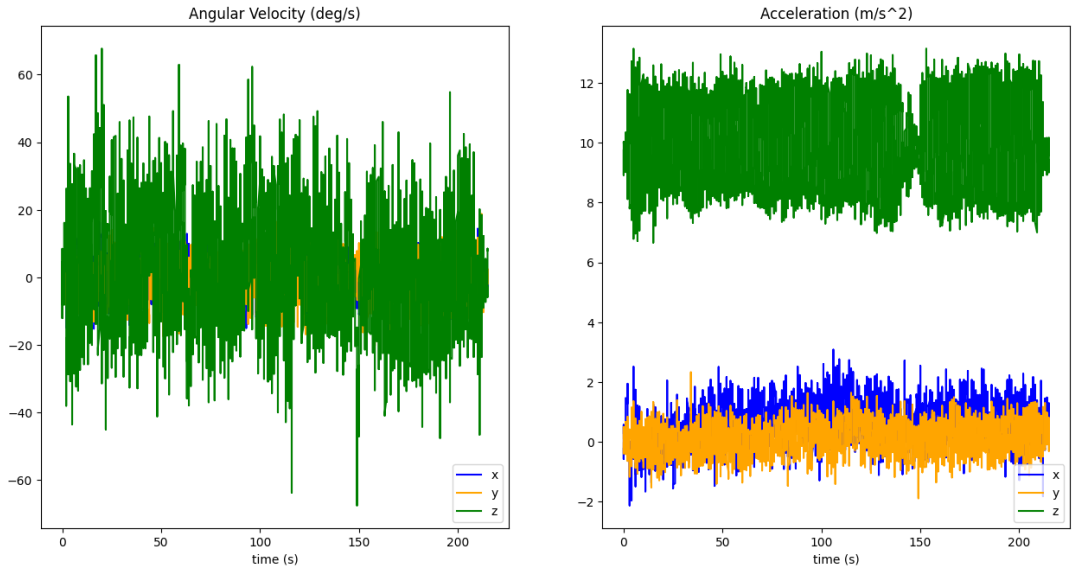

Figure 5.12: The angular velocity and acceleration in the 'Easy Maths' experiment.

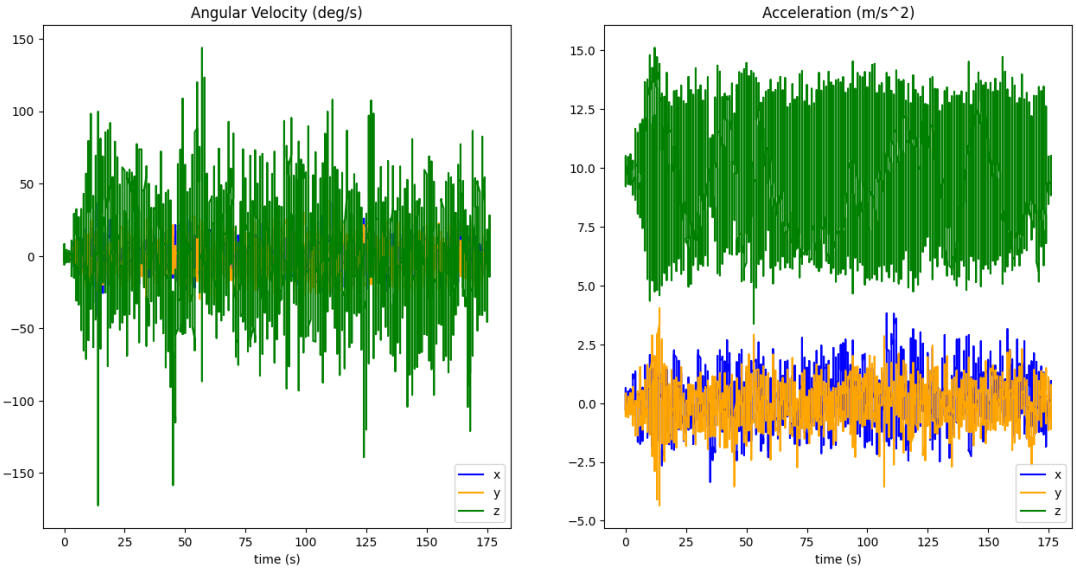

Figure 5.13: The angular velocity and acceleration in the 'Medium Maths' experiment.

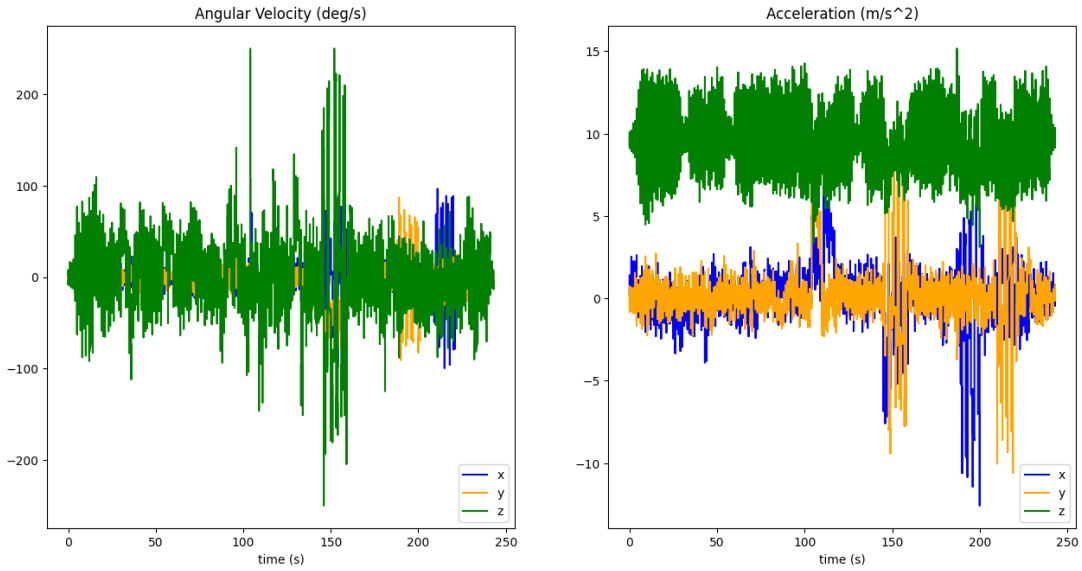

Figure 5.14: The angular velocity and acceleration in the 'Fast Maths' experiment.

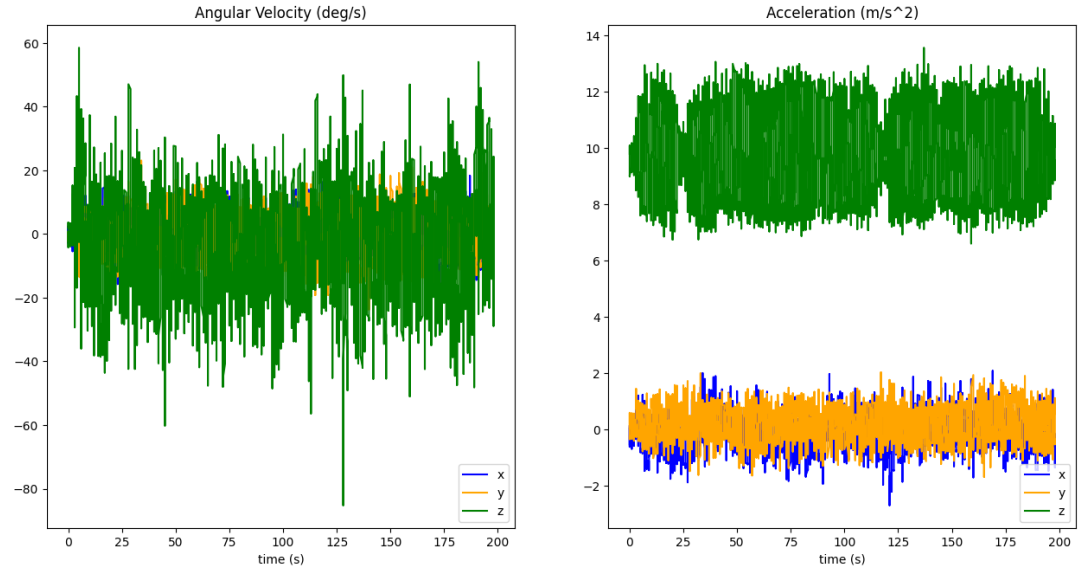

Figure 5.15: The angular velocity and acceleration in the 'Easy Quad' experiment.

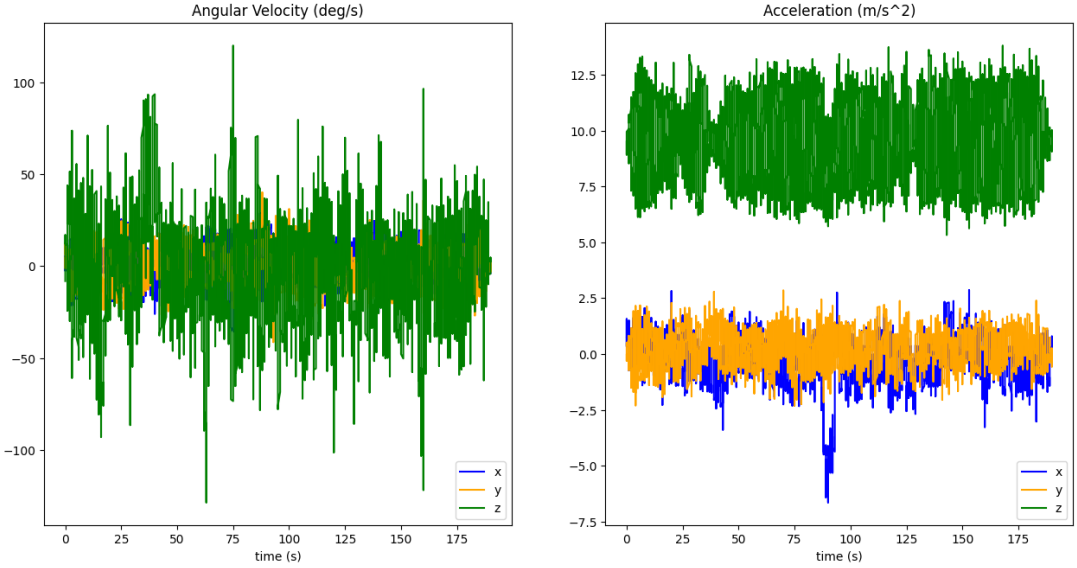

Figure 5.16: The angular velocity and acceleration in the 'Medium Quad' experiment.

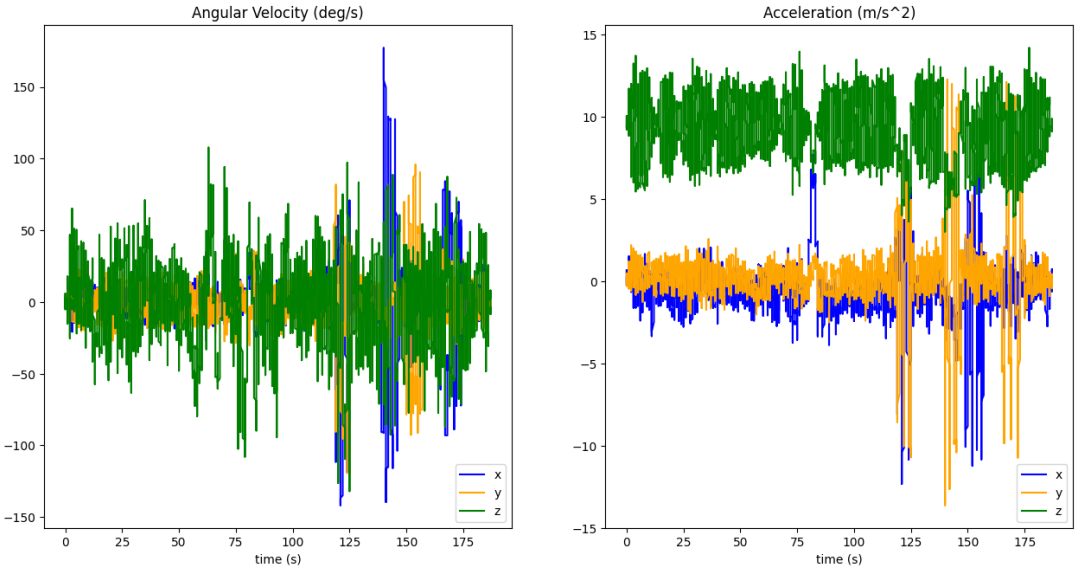

Figure 5.17: The angular velocity and acceleration in the 'Fast Quad' experiment.

5.4.1 Odometry Evaluation

Table 5.2: Comparison of ATE(Absolute Trajectory Error) and RPE(Relative Pose Error) across different datasets for FAST-LIO2 and KISS-ICP

Dataset	FAST-LIO2 ATE	FAST-LIO2 RPE	KISS-ICP ATE	KISS-ICP RPE
EASY MATHS	**0.10**	**0.40**	0.14	0.63
MEDIUM MATHS	**0.13**	**0.80**	0.21	1.27
HARD MATHS	**0.09**	**0.33**	0.21	1.30
EASY QUAD	**0.07**	**0.22**	0.09	0.28
MEDIUM QUAD	**0.07**	**0.30**	0.18	0.52
FAST QUAD	**0.05**	**0.51**	0.50	1.60

As shown in Table 5.2, FAST-LIO2 outperforms KISS-ICP for all experiments. The high-frequency output from the IMU allows for effective undistortion of the 3D scans, resulting in accurate odometry and mapping. To evaluate the system's robustness at lower frequency IMU, the IMU data in the experiments were filtered, adjusting the frequency to 50 Hz, 20 Hz, and 12.5 Hz. A lower frequency IMU could also simulate the scenario where the robot experiences more aggressive motion. As illustrated in Table 5.3, FAST-LIO2 works better for a high-frequency IMU.

Table 5.3: Comparison of ATE and RPE across different datasets and frequencies

Dataset	Frequency	ATE	RPE
EASY MATHS	50 Hz	**0.08**	**0.48**
	20 Hz	0.09	0.51
	12.5 Hz	0.09	0.50
MEDIUM MATHS	50 Hz	**0.13**	**0.91**
	20 Hz	0.15	1.00
	12.5 Hz	0.22	1.22
HARD MATHS	50 Hz	**0.06**	**0.66**
	20 Hz	0.07	0.76
	12.5 Hz	4.08	5.10
EASY QUAD	50 Hz	**0.07**	**0.24**
	20 Hz	0.07	0.25
	12.5 Hz	0.08	0.31
MEDIUM QUAD	50 Hz	**0.06**	**0.32**
	20 Hz	0.07	0.34
	12.5 Hz	0.15	0.52
FAST QUAD	50 Hz	0.08	**0.55**
	20 Hz	**0.06**	0.57
	12.5 Hz	0.17	0.78

The results indicate that reducing the IMU frequency generally degrades the performance of the odometry system, particularly in scenarios involving large rotations. At 50 Hz, the system maintains high accuracy in terms of both Absolute Trajectory Error (ATE) and Relative Pose Error (RPE). However, as the IMU frequency decreases, the system's ability to accurately estimate poses diminishes, particularly at 12.5 Hz, where significant errors are observed in the

"HARD MATHS" experiment. This degradation is due to the reduced availability of motion data, which is crucial for compensating for rapid rotations and accelerations.

The robustness of KISS-ICP's motion compensation method is crucial for accurate odometry and mapping under aggressive motions. Table 5.4 compares ATE and RPE for KISS-ICP with and without motion deskewing across different datasets.

Table 5.4: Comparison of ATE and RPE for KISS-ICP and KISS-ICP-DESKEW across different datasets

Dataset	**KISS-ICP**		**KISS-ICP-DESKEW**	
	ATE	RPE	ATE	RPE
EASY MATHS	**0.14**	**0.63**	0.14	0.63
MEDIUM MATHS	**0.21**	**1.27**	2.52	3.18
HARD MATHS	0.21	1.30	**0.15**	**1.02**
EASY QUAD	**0.09**	0.28	0.10	0.30
MEDIUM QUAD	0.18	0.52	**0.12**	**0.50**
FAST QUAD	**0.50**	**1.60**	0.73	2.01

As shown in Table 5.4, the KISS-ICP deskewing method worsens the pose estimation in the MEDIUM_MATHS experiment. The deskewing method works well in the "EASY" experiments, but as the LiDAR moves faster, the deskewing method fails to remove the distortion. This is expected as the deskewing method assumes the new pose is the same as the previous pose (constant velocity model).

5.4.2 Map Evaluation

As seen in Figure 5.18 and Figure 5.19, FAST-LIO2 and KISS-ICP produced highly accurate maps. This result suggests that the constant velocity model is sufficiently effective for motion correction and that KISS-ICP can be used in scenarios involving aggressive motion.

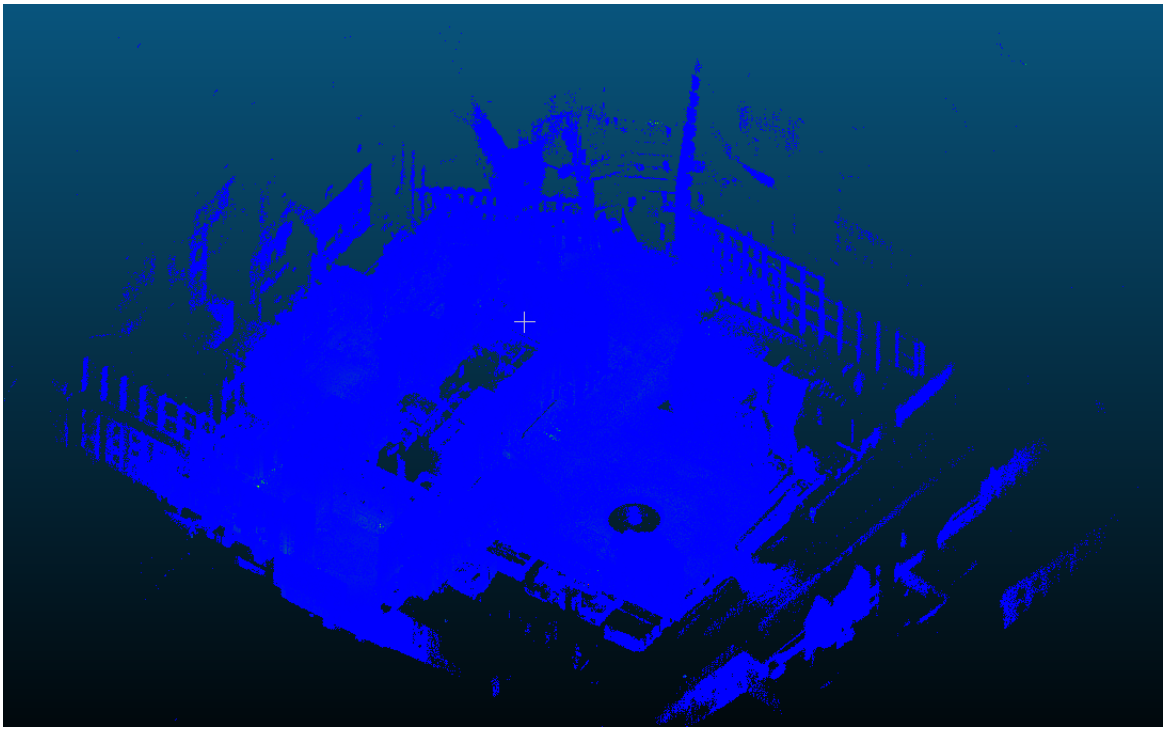

Figure 5.18: 3D map from FAST-LIO2 for the "hard maths" experiment.

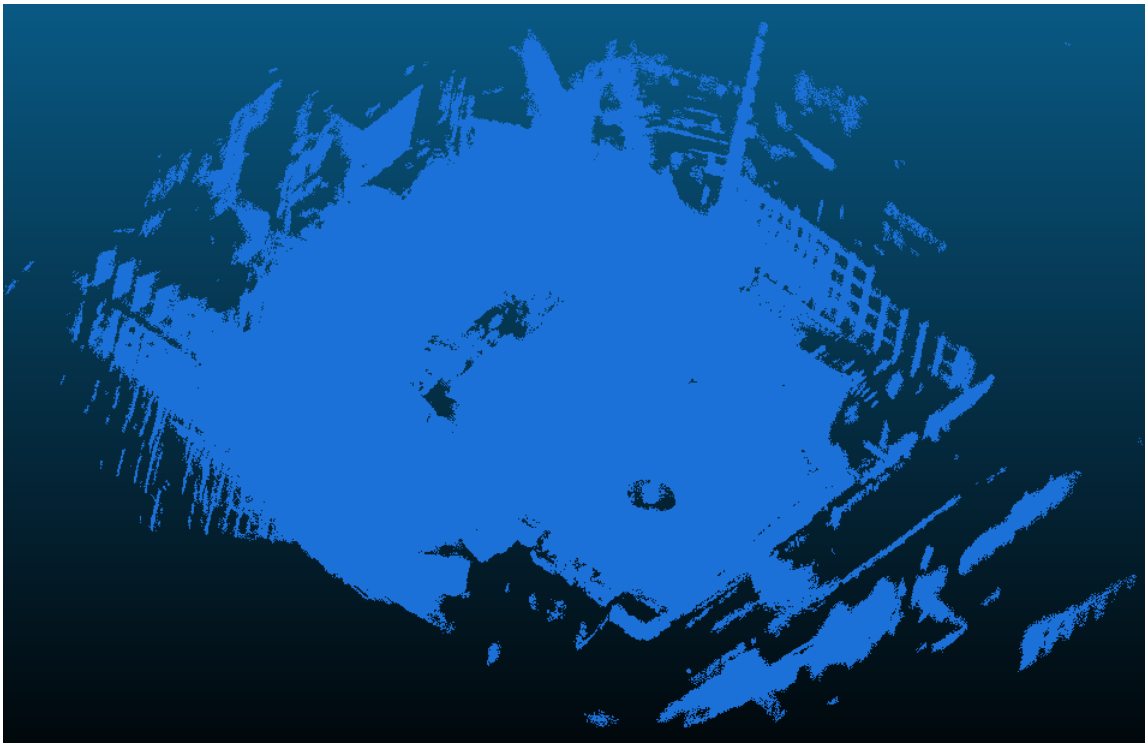

Figure 5.19: 3D map from KISS-ICP for the "hard maths" experiment.

5.5 Structured environment with large erratic motion - UCT Mechatronics Lab

The focus is now turned to create a dataset of specific experiments at the UCT Mechatronics Lab. The task of the experiments is to test the accuracy of KISS-ICP and FAST-LIO2 when the LiDAR goes through aggressive motions. A-LOAM was not evaluated as it failed to produce any meaningful results due to the physical configuration of the LiDAR used. A few experiments have been run at the University of Cape Town using an Ouster LiDAR OS2-128. The LiDAR data and IMU data are recording into PCAP files using the Ouster SDK. The PCAP files can then be converted to rosbag using [57] and [58]. The LiDAR has an integrated 9-axis IMU but only the linear accelerations and angular velocities were obtained from the Ouster SDK. The experiments included solely roll, pitch and yaw motion and a walking experiment to see how robust the method is to aggressive motions. The setup for the experiments can be found in Figure 5.20. To generate large rotations, the LiDAR is held in one hand and shaken rapidly.

The code for A-LOAM works only for Velodyne LiDARs such as VLP16, HDL-32E, and HDL-64E but the Ouster LiDARs, such as the Ouster OS2-128, have different physical configurations that are incompatible with A-LOAM. Thus the results from A-LOAM are omitted. To give an idea about the rotations, Figures 5.21, 5.24 and 5.27 show the angular velocities and accelerations during the experiments. It can be seen that the angular velocity often exceeds 200 deg/s.

The experiments carried out at UCT do not contain the ground truth transformations and thus those experiments are carried out such that the LiDAR always ends at the same position it started to move from (walking experiments) or stays at the same position (roll, pitch and yaw experiments). The parameters for KISS-ICP, adjusted to effectively handle abrupt rotations, are presented below:

KISS-ICP Parameters

```
adaptive_threshold:
  fixed_threshold: 5.0
  #initial_threshold: 5.0
  #min_motion_th: 0.5

data:
  deskew: true
  max_range: 50.0
  min_range: 0.5
  preprocess: true

mapping:
  max_points_per_voxel: 15
  voxel_size: 0.2

out_dir: results
```

fixed_threshold Value: 5.0
This parameter represents the maximum distance between corresponding points. Corresponding points larger than this parameter are ignored. A value of 5.0 is quite reasonable

in my opinion considering the aggressive motion present in the dataset. It is large enough handle significant disparities between consecutive scans.

adaptive_threshold.initial_threshold Value: 5.0
This initial threshold is set higher to allow for a broader range of potential point matches, which is beneficial for correcting large rotational differences.

adaptive_threshold.min_motion_th Value: 0.5
Raised to filter out small, irrelevant movements, helping the algorithm to focus on significant rotational changes.

data.deskew Value: true
Ensures the point cloud data is corrected for distortions caused by sensor motion, which is crucial for accurate alignment.

data.max_range Value: 50.0
Reduced to limit the maximum range of point cloud data considered, focusing on the most relevant points, hence improving computational efficiency.

data.min_range Value: 0.5
The minimum range is maintained to filter out very close points that might be unreliable.

data.preprocess Value: true
Indicates that preprocessing steps such as cropping the pointcloud frame using min_range and max_range, enhancing the input data quality.

mapping.max_points_per_voxel Value: 15
Decreased to prevent oversampling within each voxel, reducing complexity and focusing on key points for alignment.

mapping.voxel_size Value: 0.2
Increased voxel size to reduce number of point and thus reduce computational cost.

out_dir Value: results
Specifies the directory where the output results will be saved.

Figure 5.20: Experimental setup in the Mechatronics Lab at UCT, showing Ouster LiDAR (10 Hz) and IMU (100 Hz).

5.5.1 Experiment 1 (Roll)

For the roll experiment, the LiDAR was rotated only around the x-axis while standing still. Figure 5.23 shows the trajectories generated by FAST-LIO2 and KISS-ICP. FAST-LIO2 performed very well as the trajectory only slightly diverged from the origin which is expected when rotating the LiDAR manually. A side view of the 3D map generated by FASt-LIO2 is shown in Figure 5.22.

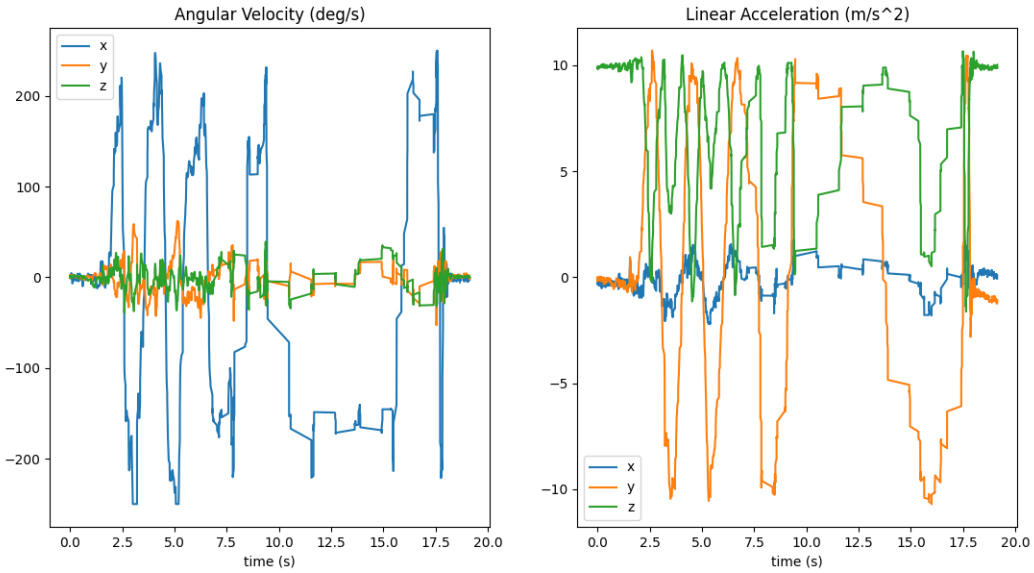

Figure 5.21: The angular velocity and acceleration in the roll experiment.

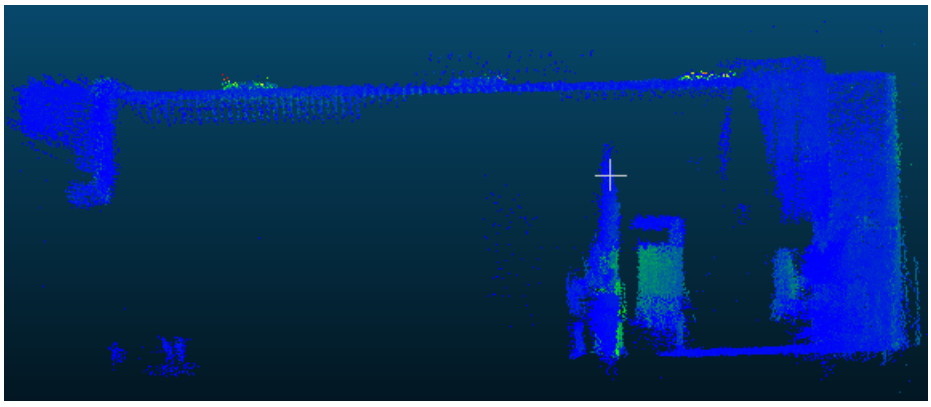

Figure 5.22: Front view of 3D map generated by FAST-LIO2 for the roll experiment.

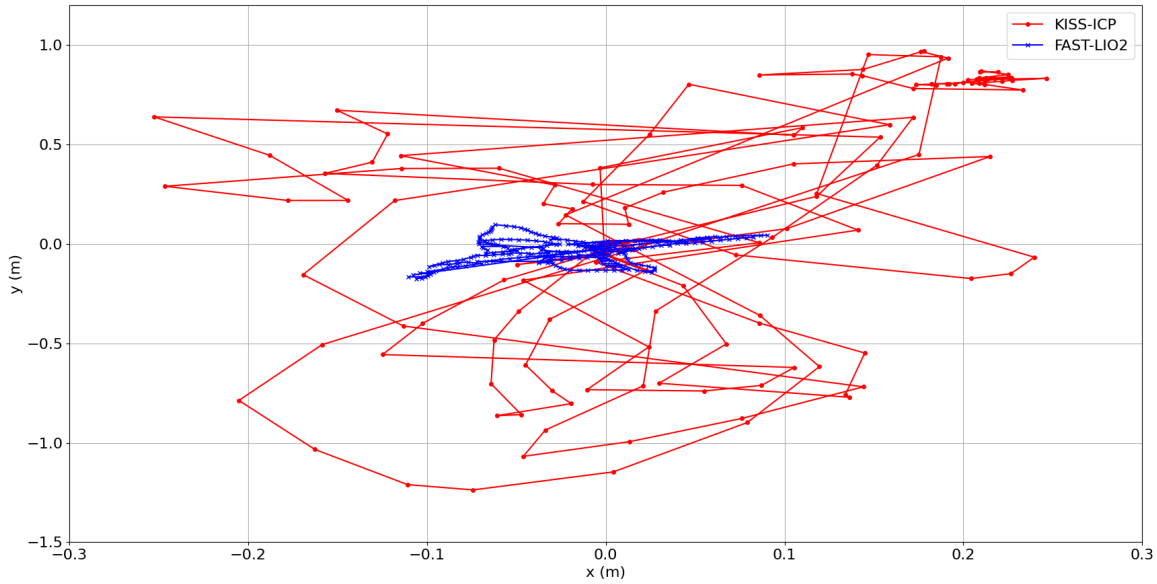

Figure 5.23: Trajectory from running FAST-LIO2 and KISS-ICP for the roll experiment.

KISS-ICP's effectiveness is limited in scenarios involving aggressive motions due to its reliance on the constant velocity model. It lacks advanced motion compensation, integration with inertial data, and robust mechanisms to handle abrupt motion and high velocities. While it performs well in stable, structured environments, its limitations become apparent in scenarios with rapid and significant motion, like a simple yaw motion only. KISS-ICP registers each scan to the local map and thus requires a dense and accurate local map to provide accurate registration. With aggressive rotation at the start of the experiment, the local map is inaccurate and so is every registration after that. In addition to this, while voxel grid filtering improves computational efficiency, it also results in the loss of fine structural details.

5.5.2 Experiment 2 (Pitch)

For the pitch experiment, the LiDAR was rotated around the y-axis while standing still. Figure 5.26 shows the trajectories generated by FAST-LIO2 and KISS-ICP. The trajectory of FAST-LIO2 moves about the origin within 0.15 m, as expected. Despite the fact that the displacements are accurate, there is a noticeable error when it comes to the orientation. This can be seen from the side view of the mapping results shown in Figure 5.25.

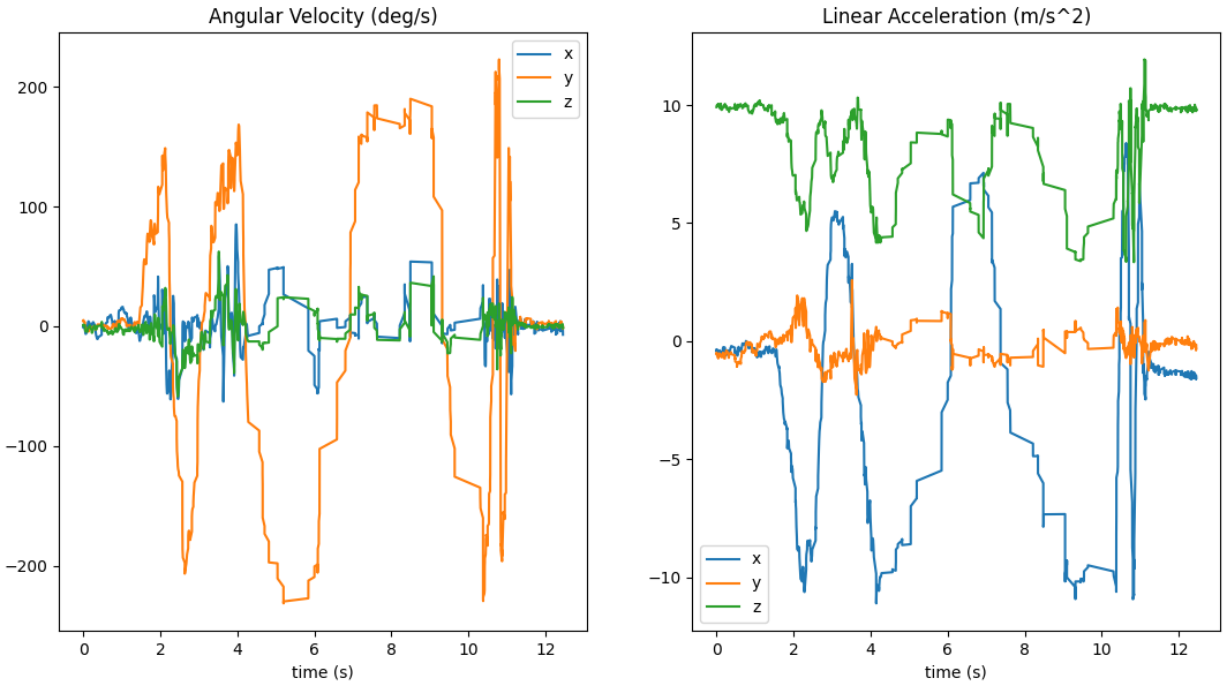

Figure 5.24: The angular velocity and acceleration in the pitch experiment.

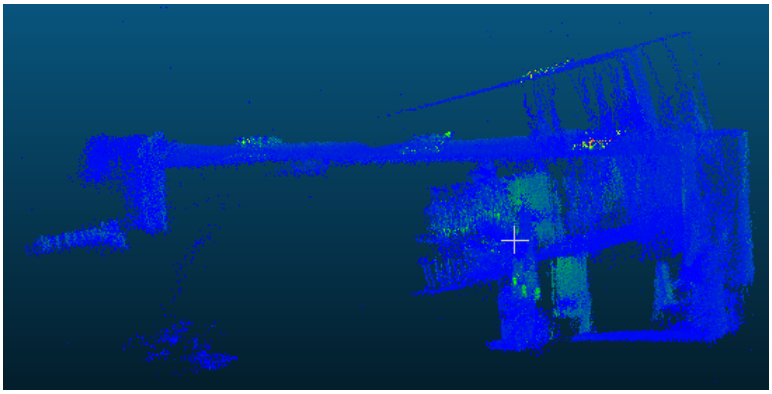

Figure 5.25: Side view of mapping results generated by FAST-LIO2 for the pitch experiment.

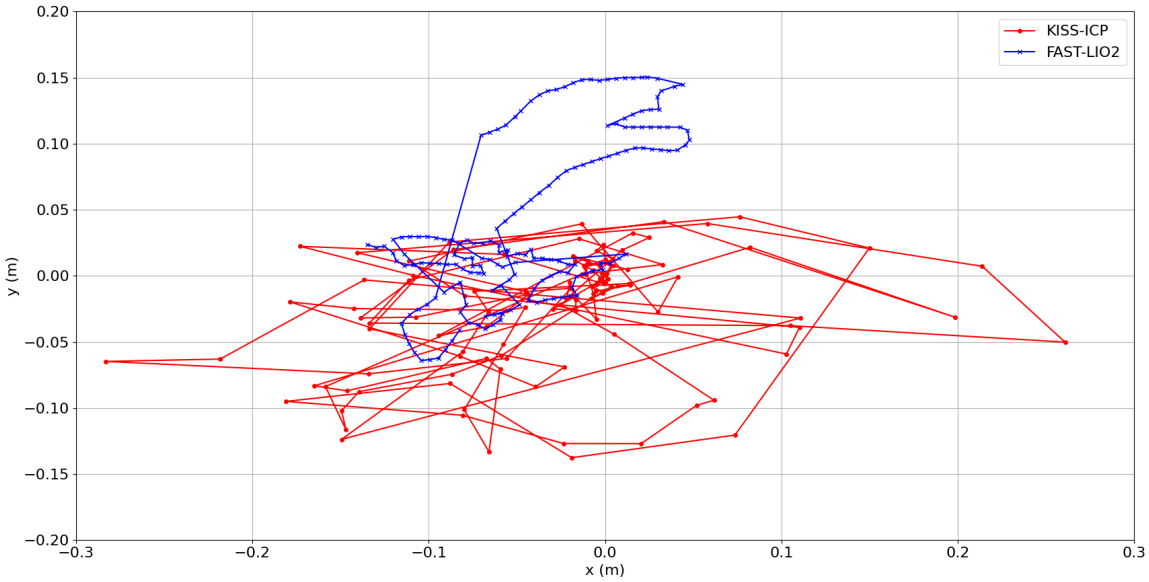

Figure 5.26: Trajectories generated by FAST-LIO2 and KISS-ICP for the pitch experiment.

In contrast to previous experiments, KISS-ICP performed surprisingly well in this scenario, demonstrating its ability to handle certain types of rotational motion effectively. FAST-LIO2 continued to demonstrate superior performance due to its inertial data integration, robust motion compensation technique, and accurate local map maintenance. This comparison underscores the importance of robust motion compensation in handling various types of aggressive rotation, reaffirming the strengths of FAST-LIO2 in challenging scenarios.

5.5.3 Experiment 3 (Yaw)

For the yaw experiment, the LiDAR was rotated around the z-axis while standing still. Figure 5.28 shows the trajectories generated by FAST-LIO2 and KISS-ICP. The trajectory for FAST-LIO2 moves about the origin within 0.15 m, as expected.

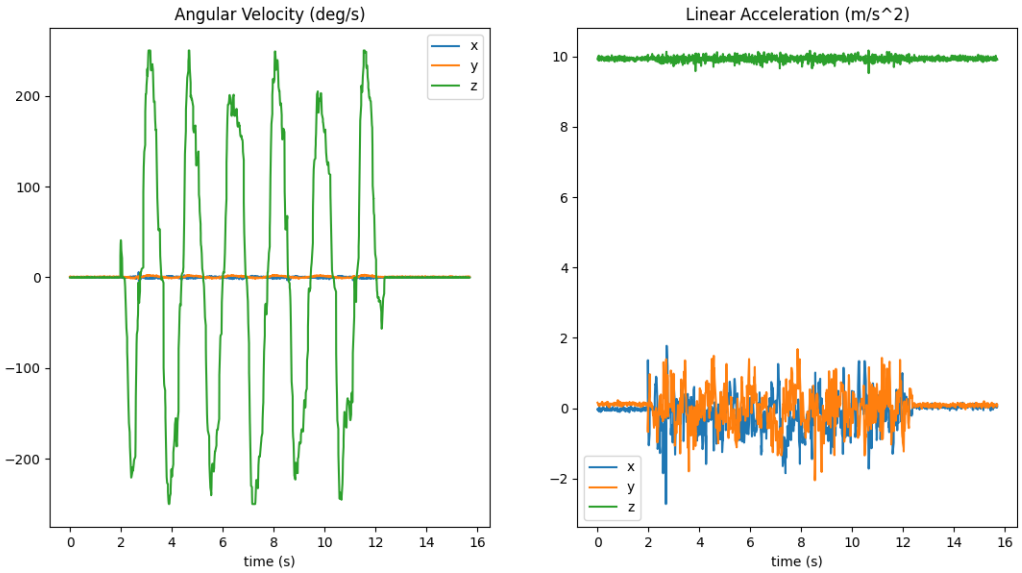

Figure 5.27: The angular velocity and acceleration in the yaw experiment.

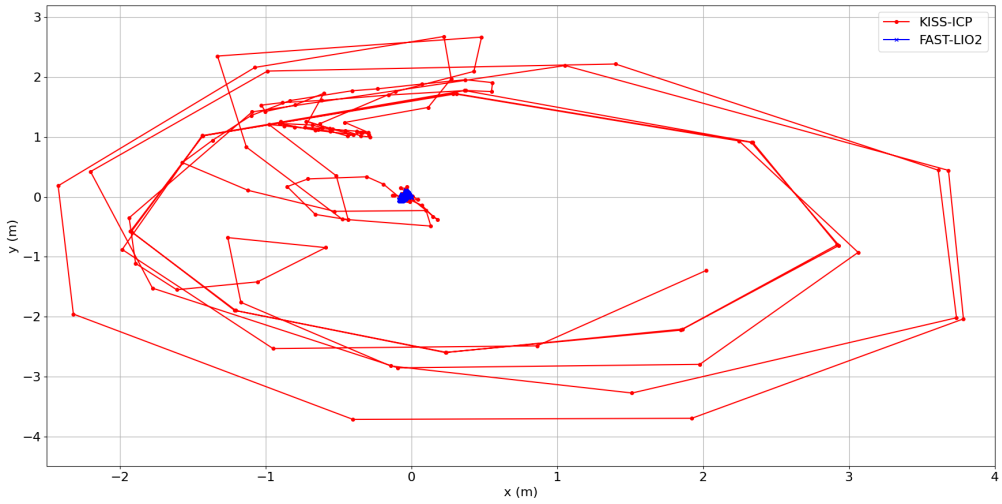

Figure 5.28: Trajectory generated by FAST-LIO2 and KISS-ICP for the yaw experiment.

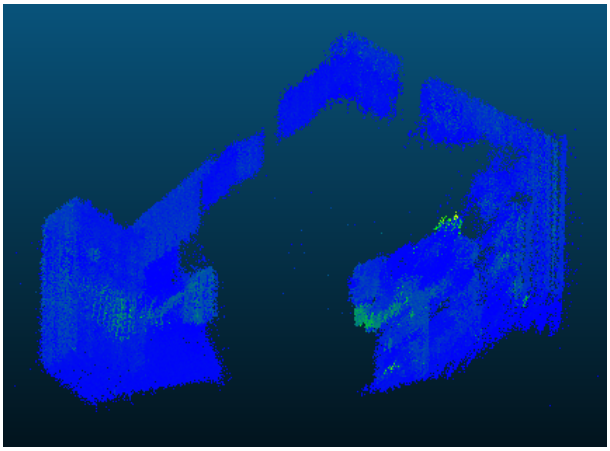

Figure 5.29: 3D map generated for the yaw experiment.

Similarly to the roll experiment, KISS-ICP proved ineffective in scenarios involving aggressive motions because of its reliance on the constant-velocity model. The resulting map did not include the ceiling and floor of the room due to the small vertical field of view (FOV) of the Ouster LiDAR. This limitation is illustrated in Figure 5.29.

5.5.4 Experiment 4 (Walking)

For the walking experiments, the LiDAR was rotated randomly around all three axes while walking in a straight line and returning to the starting position. 5.38 shows the angular velocity and linear acceleration profiles recorded during the first walking experiment. These measurements were chosen to represent challenging conditions that test the robustness and accuracy of odometry algorithms. It can be seen that the angular velocity often exceeds 200 deg/s. Ideally, the ground truth trajectory for this walking dataset should resemble a horizontal line.

Walking 1

The trajectories obtained from FAST-LIO2 and KISS-ICP are shown in Figure 5.31. FAST-LIO2 outperforms KISS-ICP in this scenario, demonstrating less drift and a straighter trajectory. Minor deviations along the y-axis are observed, likely due to the rotation and movement of the Ouster LiDAR, which affect the centre of the LiDAR system.

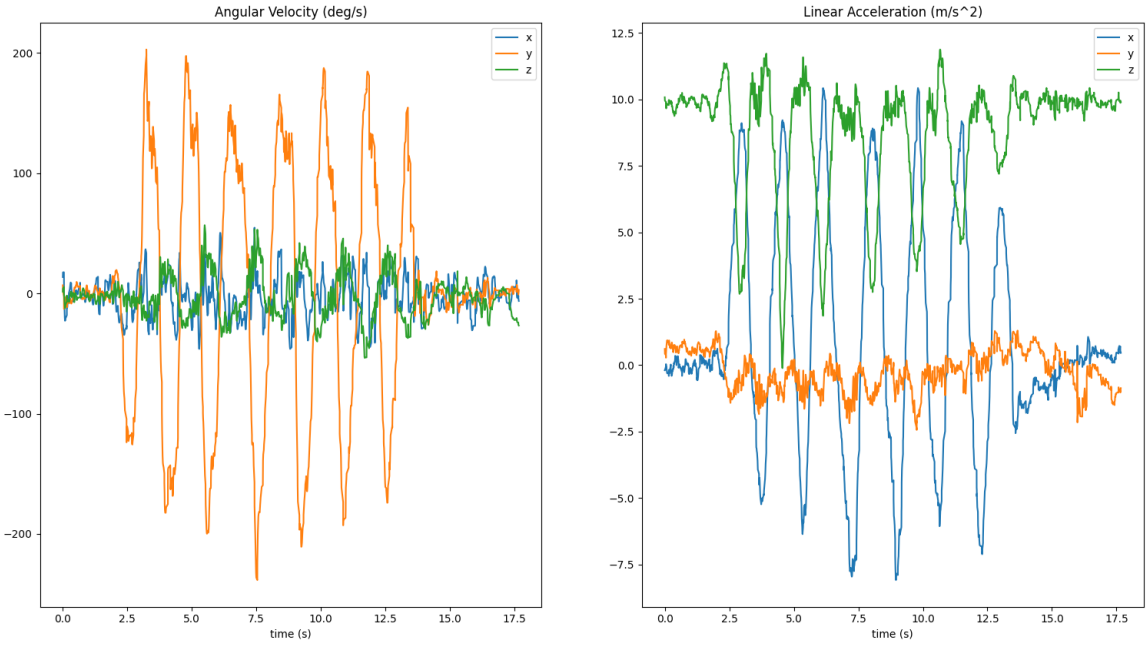

Figure 5.30: Angular velocity and linear acceleration profiles during Walking experiment 1.

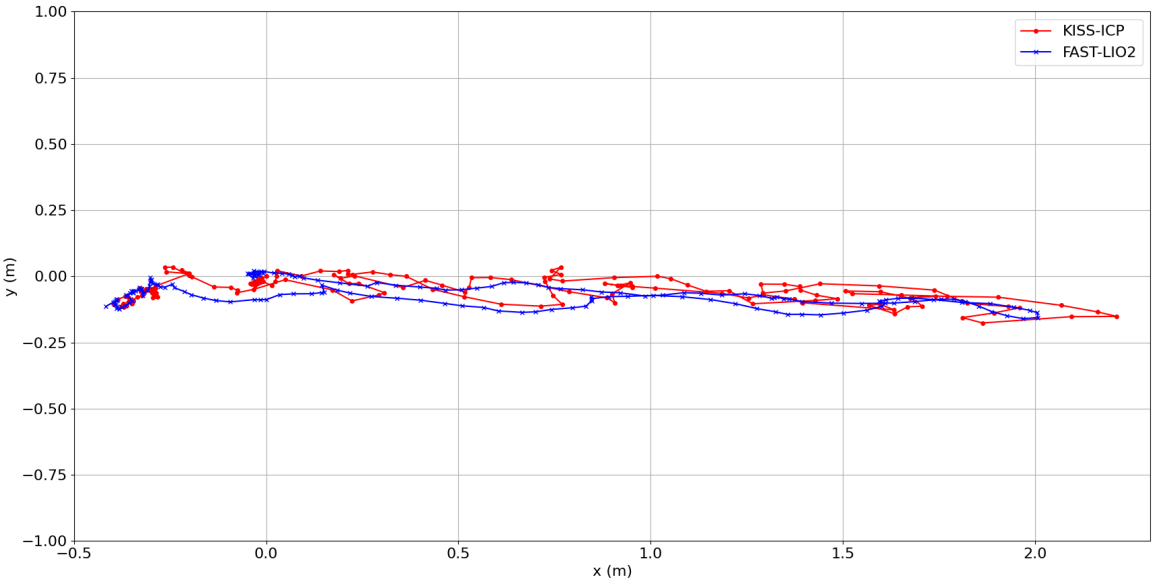

Figure 5.31: Trajectories generated by FAST-LIO2 and KISS-ICP for Walking experiment 1.

The 3D map and top view for the first walking experiment are shown in Figure 5.32 and 5.33. The top view provides a clearer representation of the map and its accuracy as we know that the top view of the room should approximate a square. The reason why FAST-LIO2 works so well is because the IMU pose at each forward propagation step is accurate enough to remove

the motion distortion in the scans.

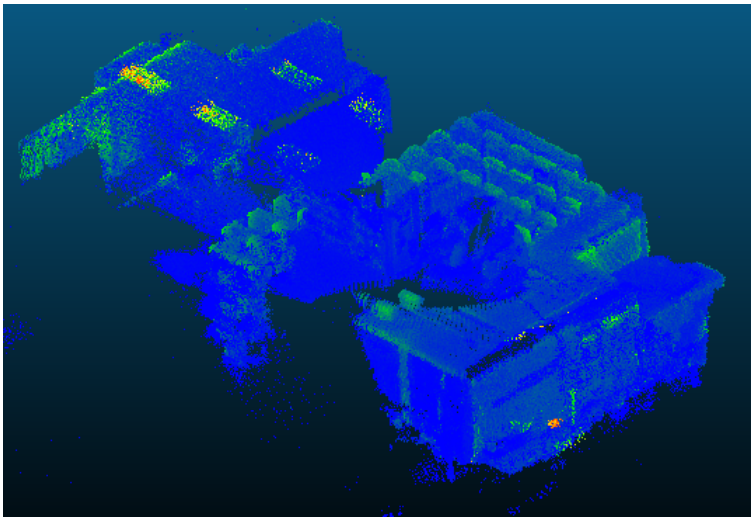

Figure 5.32: The Mapping results of FAST-LIO2 in an indoor environment with large rotation speed.

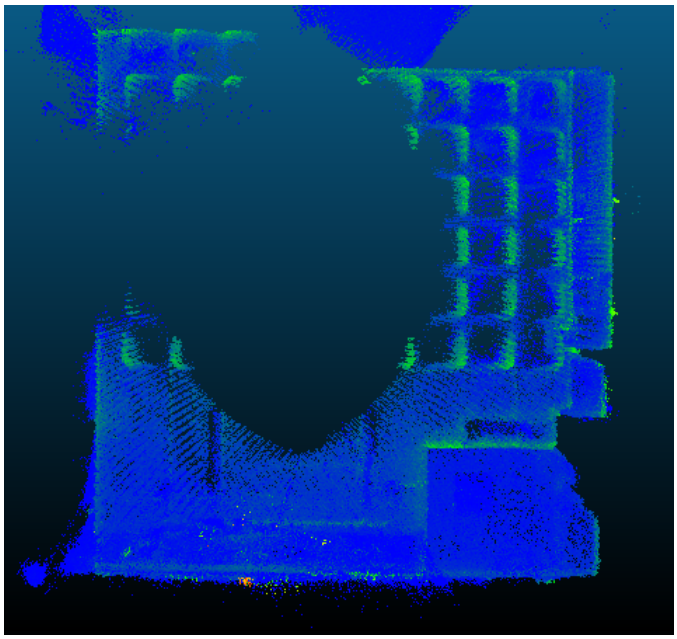

Figure 5.33: The Mapping results of FAST-LIO2 in an indoor environment with large rotation speed (Top View).

Walking 2

The trajectories for the second walking experiment generated by FAST-LIO2 and KISS-ICP are shown in Figure 5.35. It can be seen that FAST-LIO2 maintains a stable trajectory with minimal drift. The IMU-driven motion compensation method continues to effectively mitigate motion distortion, enabling accurate mapping even under aggressive rotations and movement. As in the first walking experiment, the generated map, as seen in Figures 5.36 and 5.37 is consistent with the room's layout, demonstrating FAST-LIO2's robustness in dynamic environments.

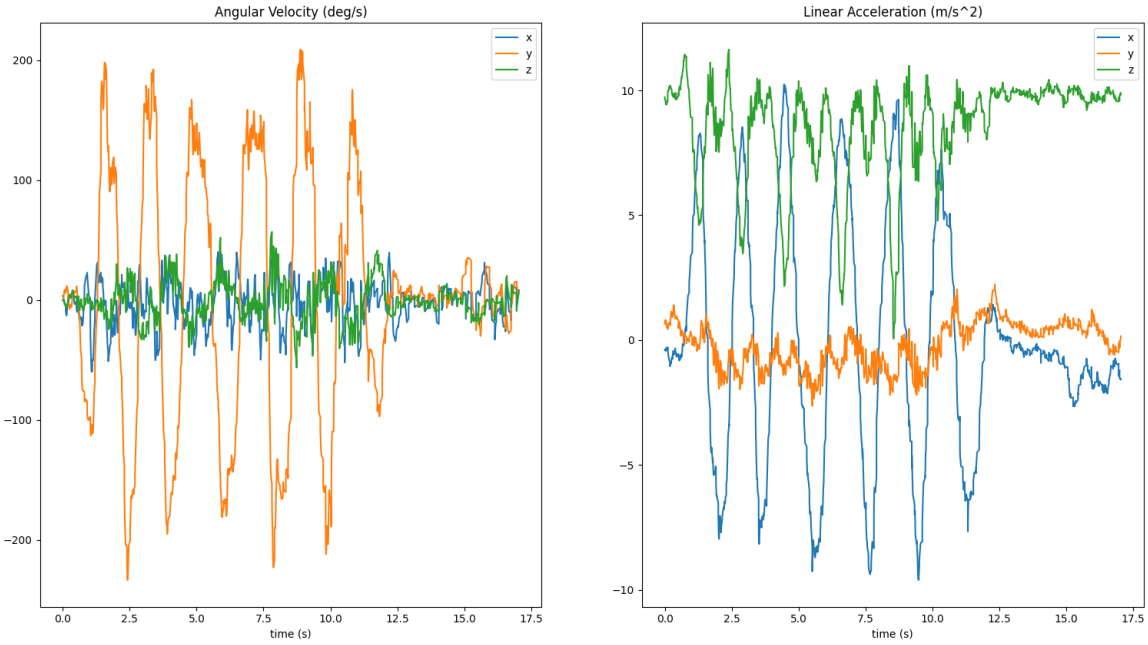

Figure 5.34: The angular velocity and acceleration in the Walking experiment 2.

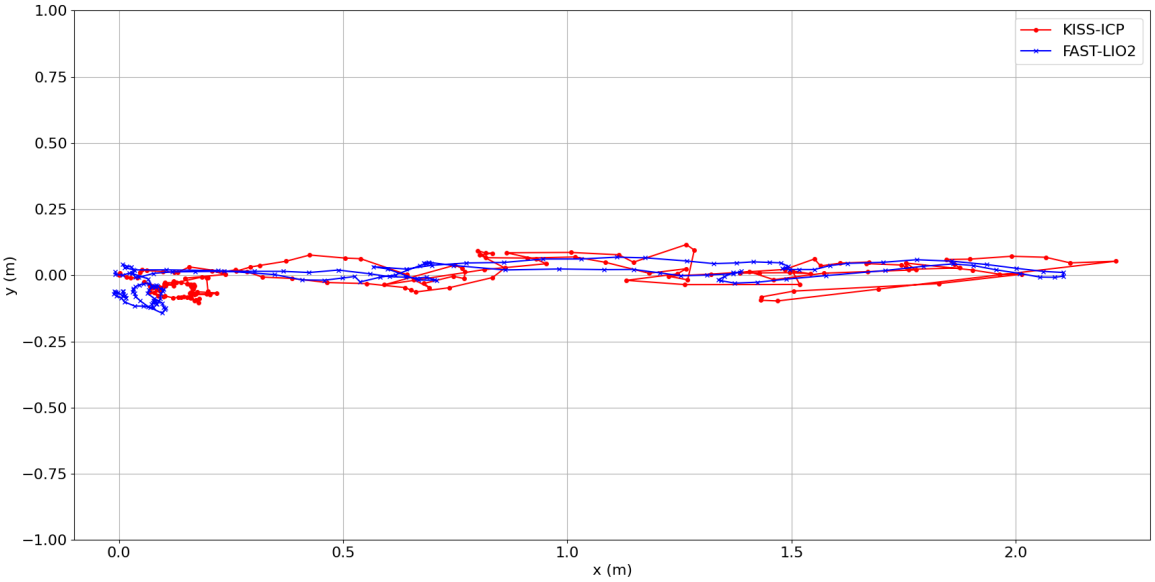

Figure 5.35: Trajectories generated by FAST-LIO2 and KISS-ICP for Walking experiment 2.

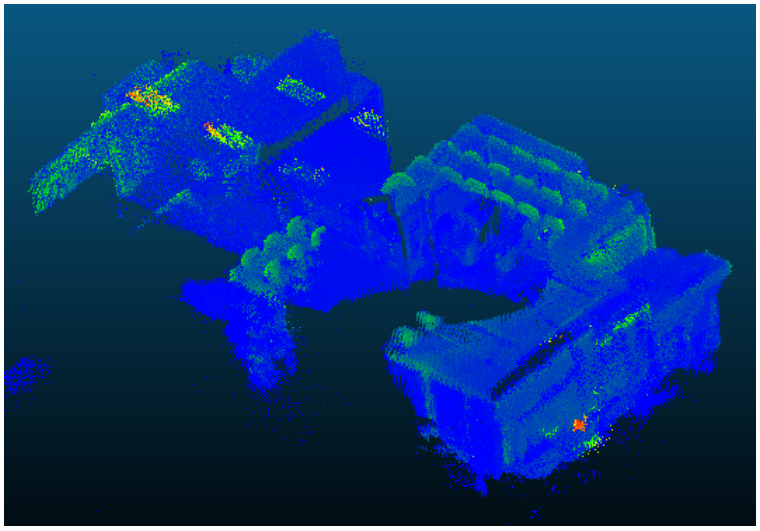

Figure 5.36: 3D map generated by FAST-LIO2 for Walking experiment 2.

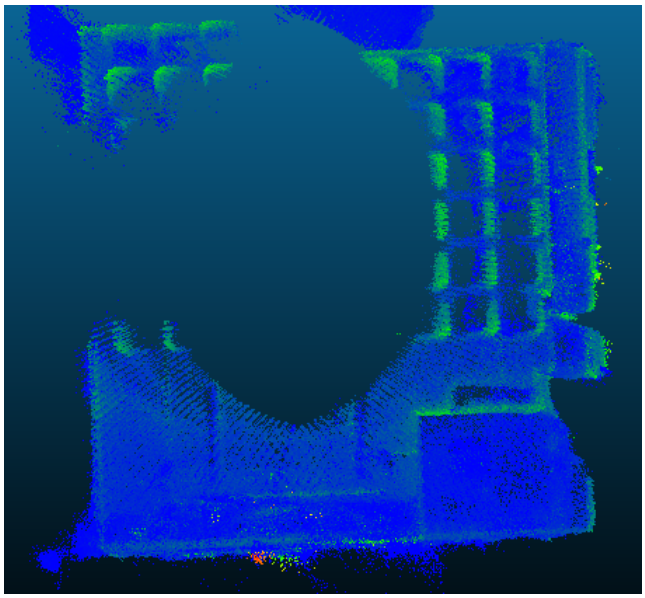

Figure 5.37: Top view of map generated by FAST-LIO2 for Walking experiment 2.

KISS-ICP appears to perform well in this walking experiment. The low deviations and generally straight trajectory indicate that KISS-ICP accurately estimated the path, with minimal drift. This suggests that, in this particular scenario, KISS-ICP's scan-matching and deskewing were sufficient to handle the motion, possibly due to stable and consistent LiDAR readings. Factors such as the absence of abrupt rotations or challenging environmental features likely contributed to its stability in this instance.

Walking 3

The trajectories generated by FAST-LIO2 and KISS-ICP in the third walking experiment are shown in Figure 5.39. Thanks to the IMU integration and EKF-based motion compensation, FAST-LIO2 maintained a stable trajectory with minimal drift, even when the LiDAR was

exposed to frequent rotations and high angular speeds. Its ability to account for both translation and rotation at a high frequency is a key reason it performed well in this scenario.

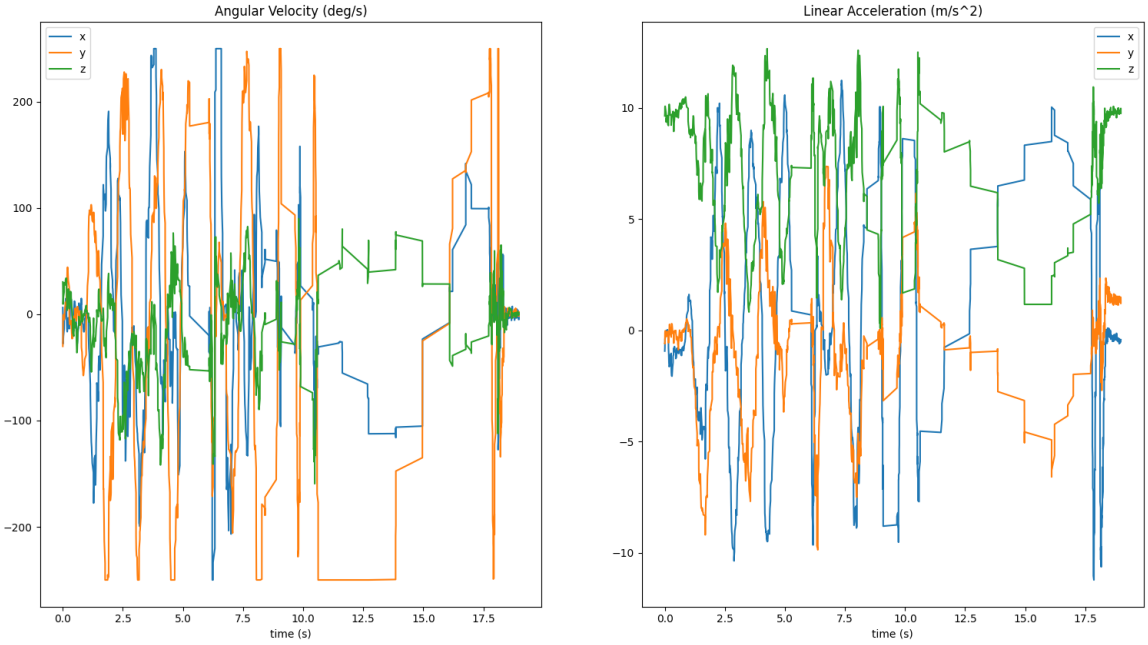

Figure 5.38: The angular velocity and acceleration in Walking experiment 3.

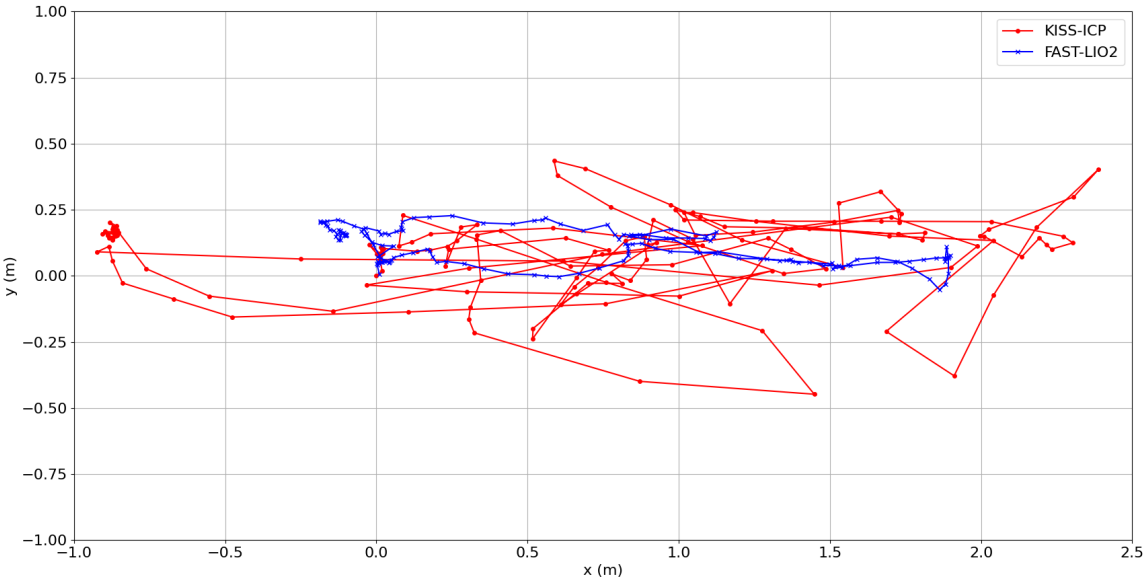

Figure 5.39: Trajectories generated by FAST-LIO2 and KISS-ICP for Walking experiment 3.

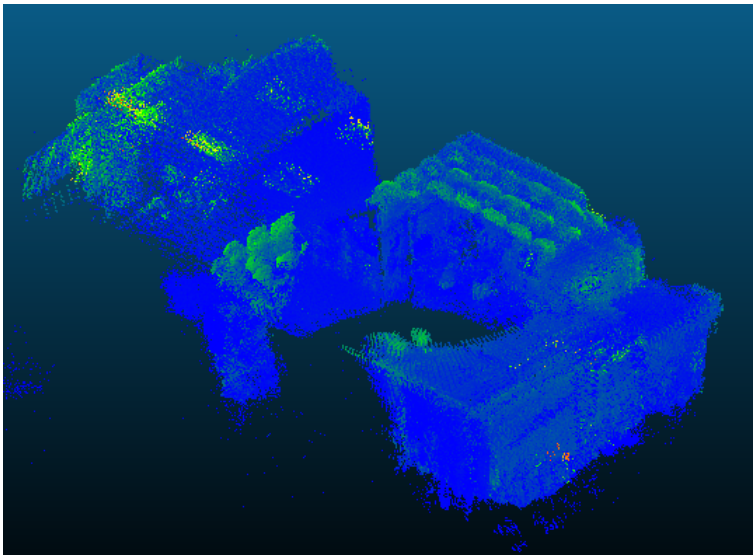

Figure 5.40: 3D map generated by FAST-LIO2 during Walking experiment 3.

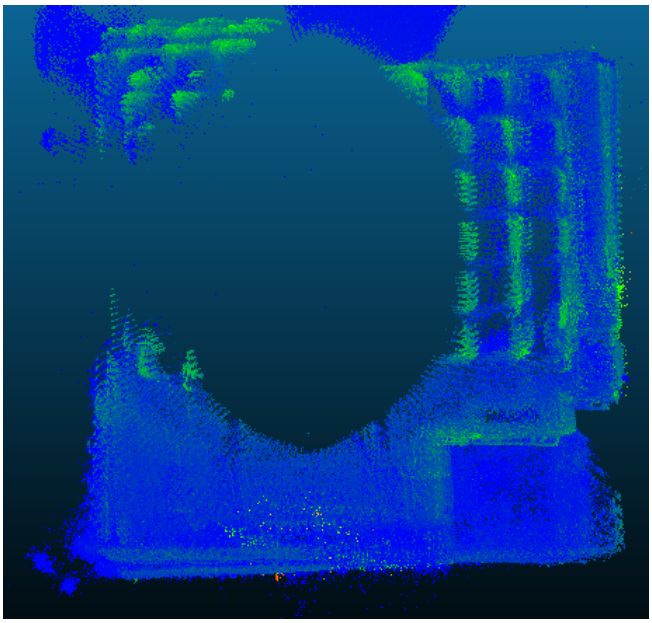

Figure 5.41: Top view of map generated by FAST-LIO2 during Walking experiment 3.

Due to its purely LiDAR-based approach, KISS-ICP struggled to accurately capture the LiDAR's true trajectory during the walking experiment. The frequent rotations led to motion distortion that it could not compensate for, resulting in noticeable drift and deviations from the actual trajectory. The lack of IMU data means that it has no mechanism to predict motion or handle high-speed rotations effectively, which led to less accurate odometry in this case. The map illustrates the ability of FAST-LIO2 to produce accurate maps even when handling motion distortions, as shown in Figures 5.40 and 5.41.

5.6 Ablation Studies

5.6.1 Motion Compensation

The most important process in a LiDAR odometry system for a robot that undergoes aggressive motion is motion compensation. To assess the effectiveness of the motion compensation method, KISS-ICP and FAST-LIO2 are implemented with and without any compensation applied and the results are compared.

In LiDAR odometry systems, motion compensation is a critical process, especially for robots that undergo aggressive or rapid movements. Motion compensation corrects for distortions caused by the robot's motion during LiDAR scanning, allowing for more accurate pose estimation. This ablation study evaluates the effectiveness of motion compensation by implementing KISS-ICP and FAST-LIO2 with and without compensation. The aim is to determine whether compensating for motion during scans yields significant improvements in accuracy.

The following KISS-ICP parameters were optimised for indoor environments:

KISS-ICP Parameters

```
adaptive_threshold:
  fixed_threshold: 5.0
  initial_threshold: 5.0
  min_motion_th: 0.5

data:
  deskew: true/false
  max_range: 50.0
  min_range: 0.5
  preprocess: true

mapping:
  max_points_per_voxel: 15
  voxel_size: 0.2

out_dir: results
```

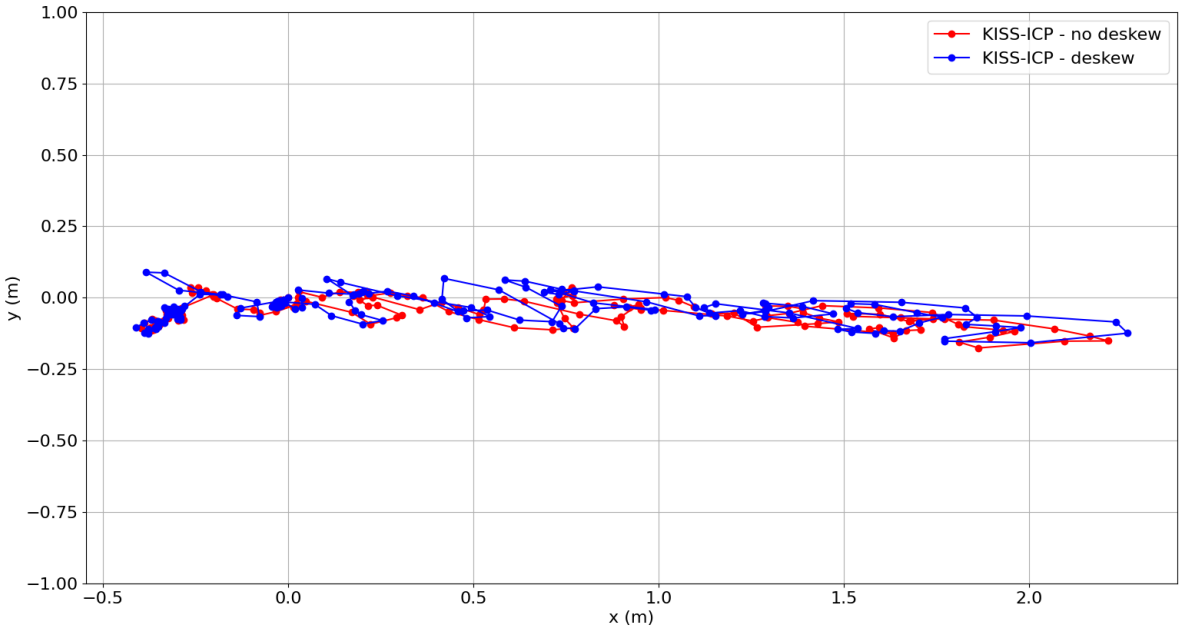

Figure 5.42: Trajectory results generated by FAST-LIO2 and KISS-ICP for Walking dataset 1 with and without motion compensation applied.

Figures 5.42, 5.43 and 5.44 illustrate the trajectory results for KISS-ICP across three walking datasets with and without motion compensation. Enabling motion compensation in KISS-ICP did not yield significant accuracy improvements in the first two walking datasets. This is attributed to the relatively steady motion during the initial scans, which allowed KISS-ICP to build a dense local map for subsequent registration.

KISS-ICP's motion compensation relies on a constant velocity model, which assumes the robot maintains a steady speed during the scan. In the third walking experiment, this assumption did not hold due to the rapid motion of the LiDAR. As a result, applying deskew in KISS-ICP led to poorer trajectory accuracy compared to the non-deskewed version.

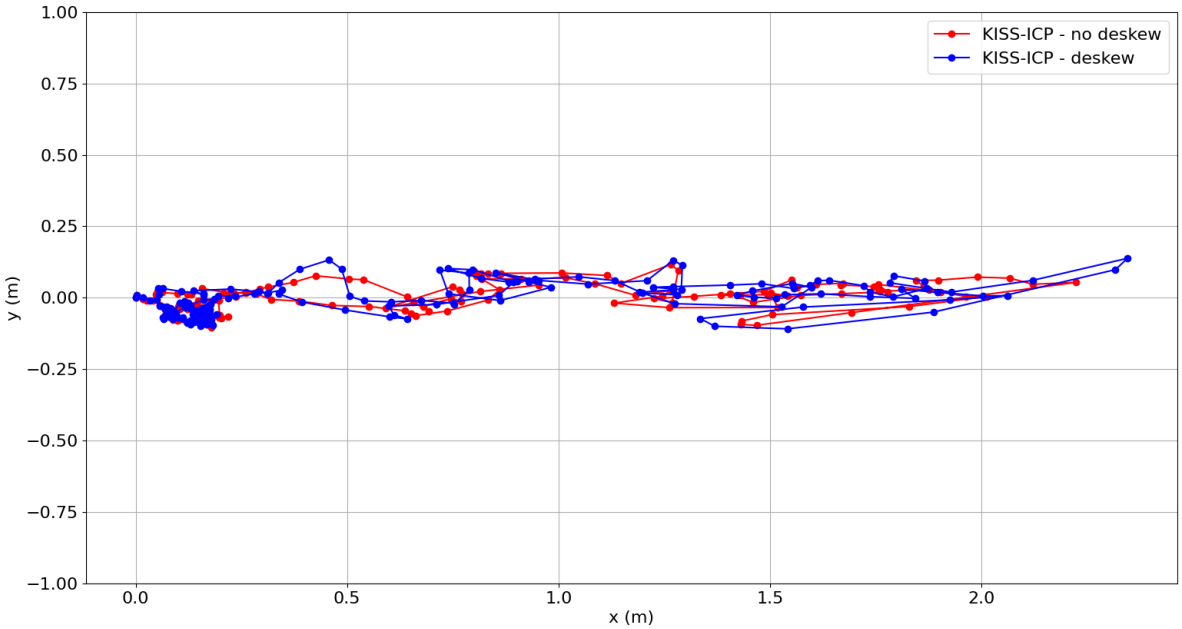

Figure 5.43: Trajectory results generated by FAST-LIO2 and KISS-ICP for Walking dataset 2 with and without motion compensation applied.

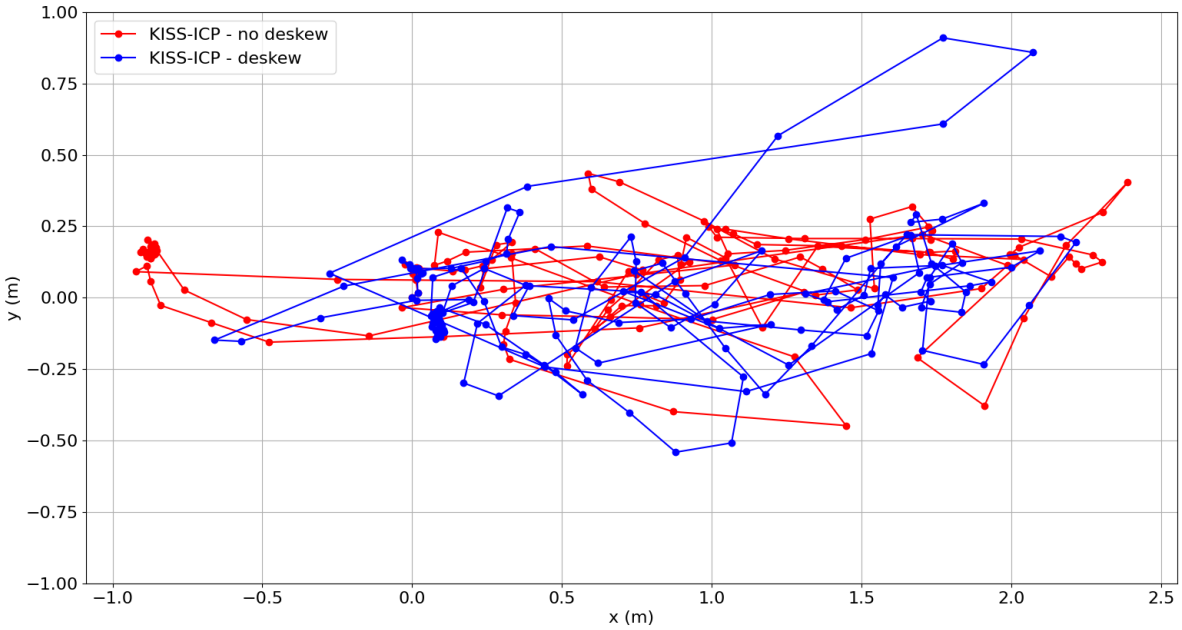

Figure 5.44: Trajectory results generated by FAST-LIO2 and KISS-ICP for Walking dataset 3 with and without motion compensation applied.

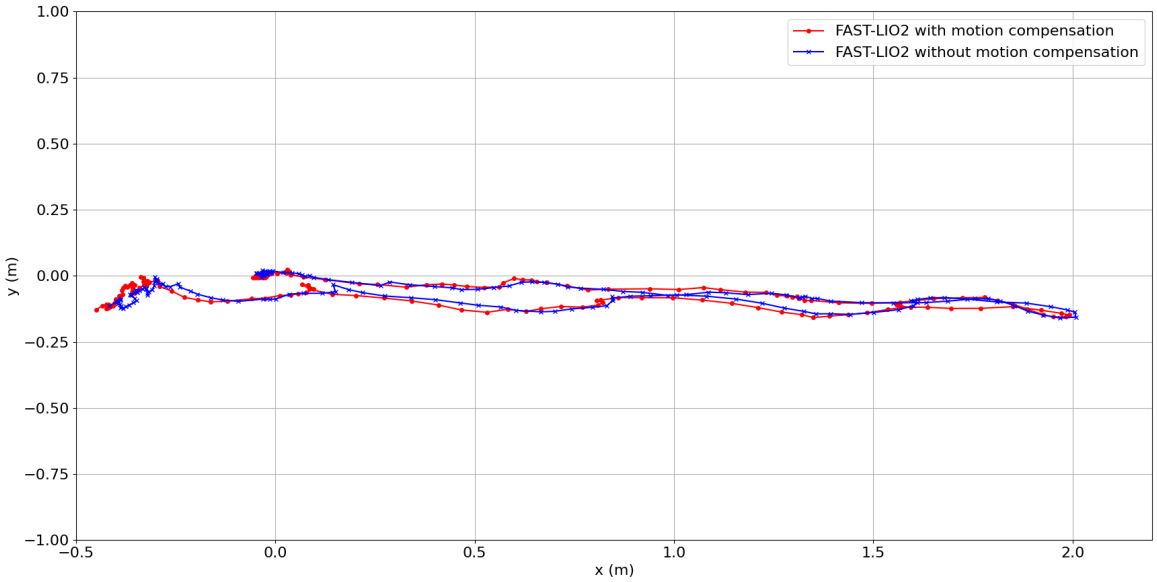

Figure 5.45: Comparison results between FAST LIO with no motion compensation applied and motion compensation applied for Walking 1 experiment.

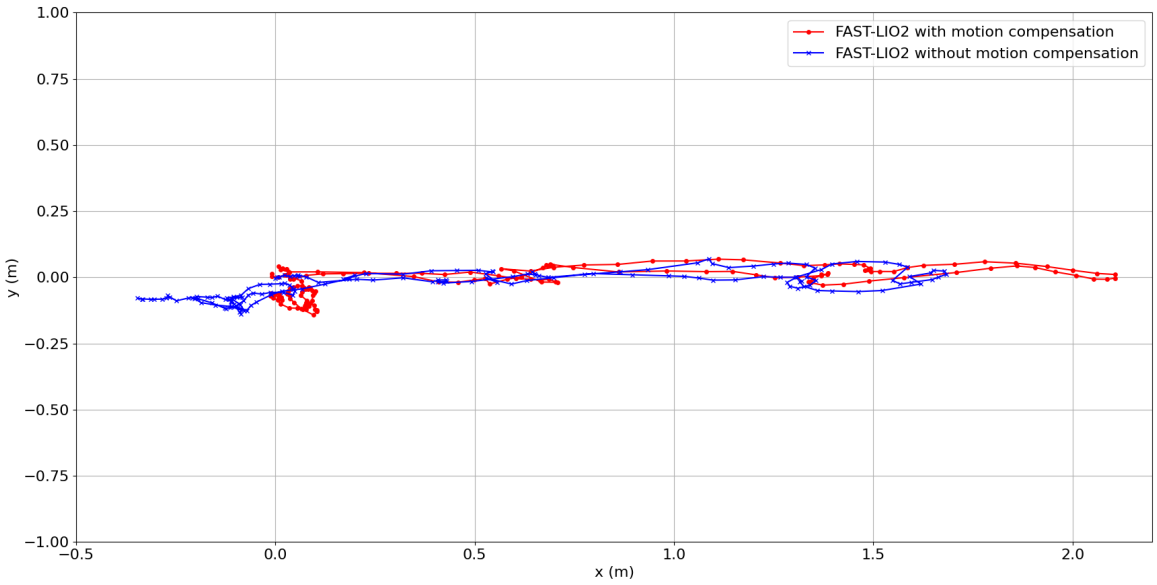

Figure 5.46: Comparison results between FAST LIO with no motion compensation applied and motion compensation applied for Walking 2 experiment.

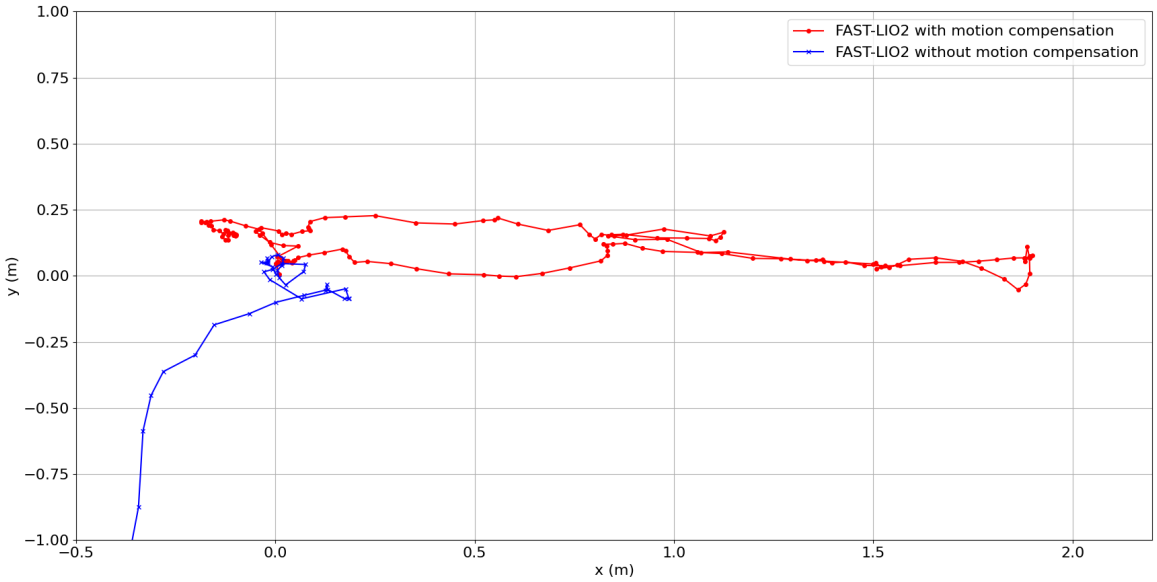

Figure 5.47: Comparison results between FAST LIO with no motion compensation applied and motion compensation applied for Walking 3 experiment.

Figures 5.45, 5.46 and 5.47 show the trajectories for FAST-LIO2 comparing configurations with and without motion compensation. The results indicate that while motion compensation has minimal impact on the first two walking experiments, the trajectory in the third walking experiment diverges significantly without compensation. This divergence is likely due to the high angular velocities across all three axes in the third experiment, as opposed to predominantly y-axis motion in the first two experiments.

FAST-LIO2 demonstrates robustness under aggressive motion because it integrates high-frequency IMU data to continuously predict the robot's pose between LiDAR scans. Using linear acceleration and angular velocity measurements, FAST-LIO2 captures rapid motion changes between scans, allowing for more accurate pose estimation. This system also uses a technique called backward propagation to correct for motion distortion. Backward propagation transforms each LiDAR point to a common reference frame, usually the end of the scan, ensuring accurate point alignment and minimising errors in pose estimation and map construction.

The results suggest that FAST-LIO2, with its continuous integration of IMU data and backward propagation for motion compensation, is more suitable for applications involving aggressive or high-speed motion. Conversely, KISS-ICP performs adequately in scenarios with steady or limited motion but lacks the continuous integration that would allow it to handle rapid motion effectively.

5.6.2 Features vs Points

This ablation study examines the impact of using feature points versus raw points for odometry accuracy in FAST-LIO2. Traditionally, feature extraction helps isolate distinctive geometric structures (such as edges and planes) that serve as reliable landmarks in the environment. However, FAST-LIO2 removes the explicit feature extraction process to increase system flexibility across different sensor types and environmental conditions. This configuration allows the algorithm to operate on raw points alone, thereby increasing processing speed.

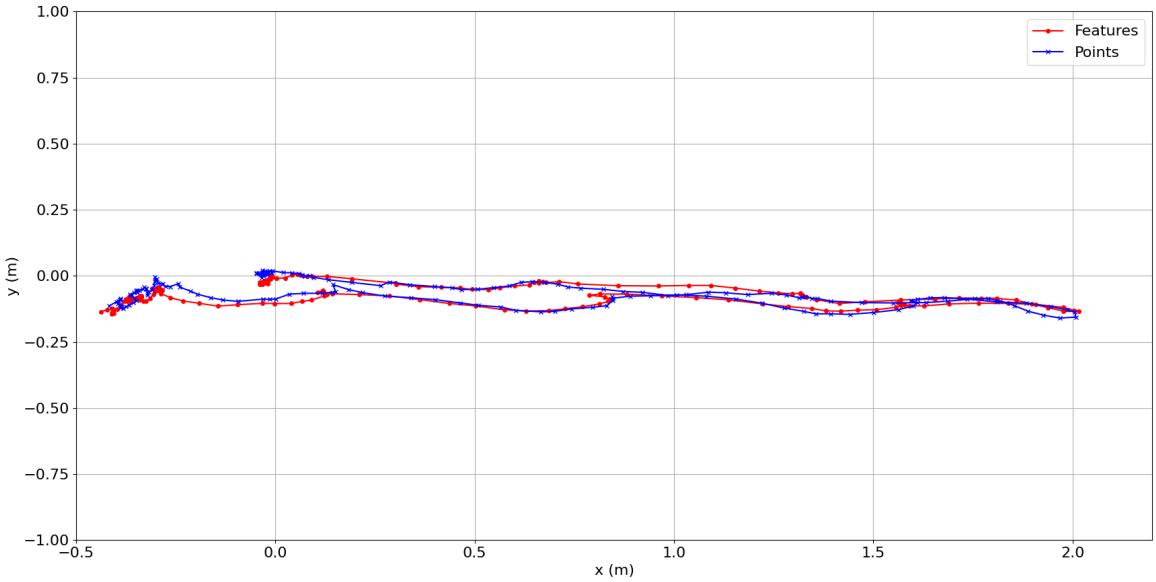

Figure 5.48: Comparison results between FAST LIO with feature extraction enabled and disabled for Walking Experiment 1.

From Figure 5.45, the trajectories are close to each other, but the red trajectory (with feature extraction) shows slightly less divergence and smoother transitions, indicating better performance in handling aggressive motion.

Figure 5.48 illustrates the trajectories for Walking Experiment 1 with and without feature extraction enabled. Although both configurations yield closely aligned trajectories, the red trajectory (with feature extraction) shows marginally smoother transitions, suggesting that feature extraction provides enhanced stability under aggressive motion.

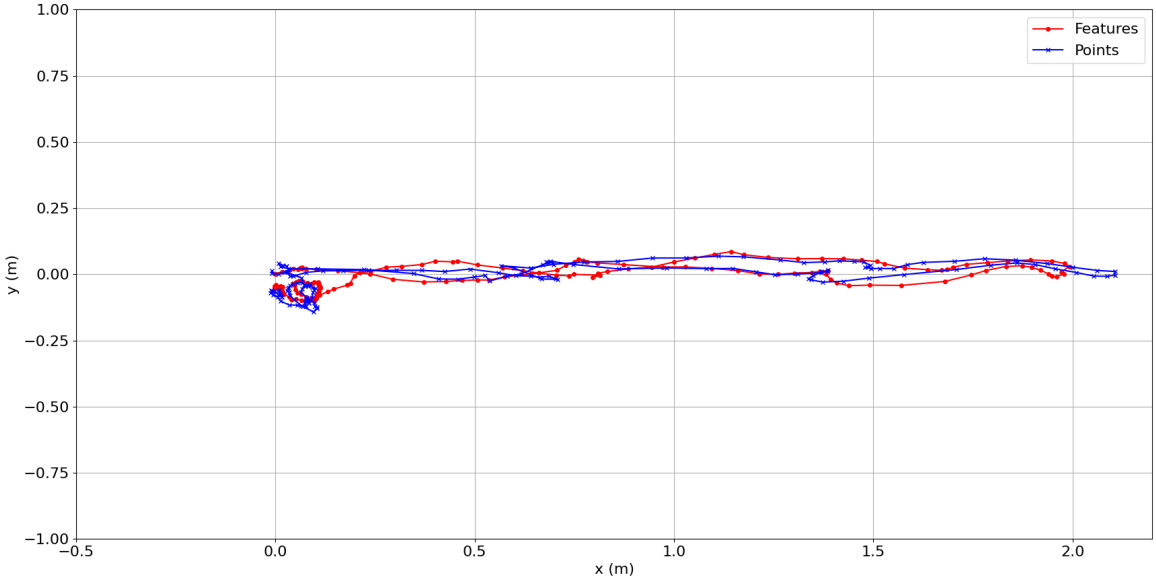

Figure 5.49: Comparison results between FAST-LIO2 with feature extraction enabled and disabled for Walking Experiment 2.

Figure 5.49 shows the results for the Walking Experiment 2. Similar to the first experiment, the trajectory generated with feature extraction exhibits greater stability, particularly during rotations. Feature extraction appears to handle aggressive rotations more effectively, resulting in a trajectory with fewer minor deviations.

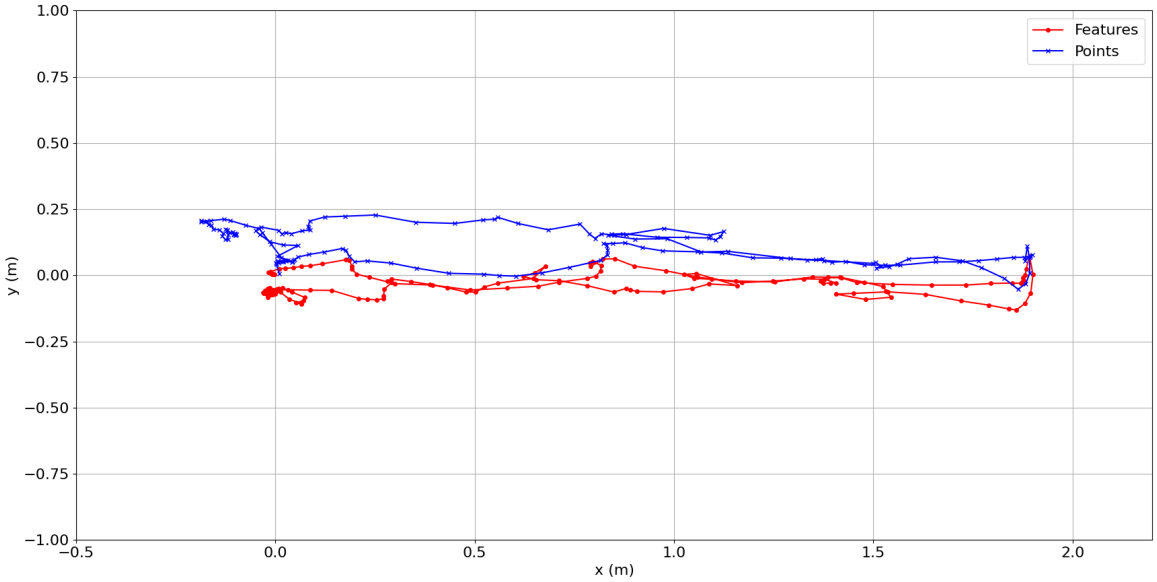

Figure 5.50: Comparison results between FAST-LIO2 with feature extraction enabled and disabled for Walking Experiment 3.

In Figure 5.50 for Walking Experiment 3, the difference between the two configurations becomes more pronounced. Here, the red trajectory (with feature extraction) remains closer to the true path, whereas the blue trajectory (without feature extraction) diverges more significantly. This divergence highlights that the absence of feature extraction is more impactful in scenarios with rapid motion.

It is evident that FAST-LIO2 with feature extraction enabled outperforms the configuration without feature extraction under aggressive motion conditions. The use of feature points provides a more stable and reliable set of correspondences, improving the accuracy and robustness of the odometry system. In scenarios with high rotational dynamics, this distinction becomes even more crucial, as seen in the provided plots. A major factor in feature-wise matching outperforming points-matching is that indoor environments, such as labs, have numerous edges and planes due to walls, furniture, and other objects. These features are easily detectable and provide strong geometric constraints for scan matching. The high number of feature points in such environments contributes to a better local map and more accurate matching. The LiDAR's vertical field of view (FOV) of 22.5° is well-suited to capture the vertical structures common in indoor environments, further enhancing the quality of feature extraction.

5.7 UCT Zoo Dataset

5.7.1 Husky Platform

Data was gathered at the University of Cape Town's Old Zoo for a real-life experiment in an outdoor environment with uneven terrain. The zoo is approximately 2 hectares, consisting mainly of large trees, small vegetation, and a few abandoned semi-structured buildings and animal enclosures. A Clearpath Husky UGV fitted with a stereo ZED 2i camera system and a Velodyne HDL-32E laser scanner, shown in Figure 5.51, was used as the experimental platform.

Figure 5.51: Photo of the Clearpath Husky along with sensors used for data collection at the UCT Old Zoo area.

The task of this experiment is to test the performance of LOAM in an outdoor environment without any motion compensation and IMU. To demonstrate the performance of A-LOAM in outdoor environments, five different "runs", A, B, C, D and E, were recorded in rosbags using ROS and and these were then inserted in the A-LOAM pipeline. The odometry results from the experiments are shown in Figure 5.52. In all the experiments, the husky returns to the starting point after travelling over 200 m. There is little to no drift in all experiments. This shows that A-LOAM can be used to create highly accurate 3D maps of an outdoor environment without any deskewing or loop closing techniques.

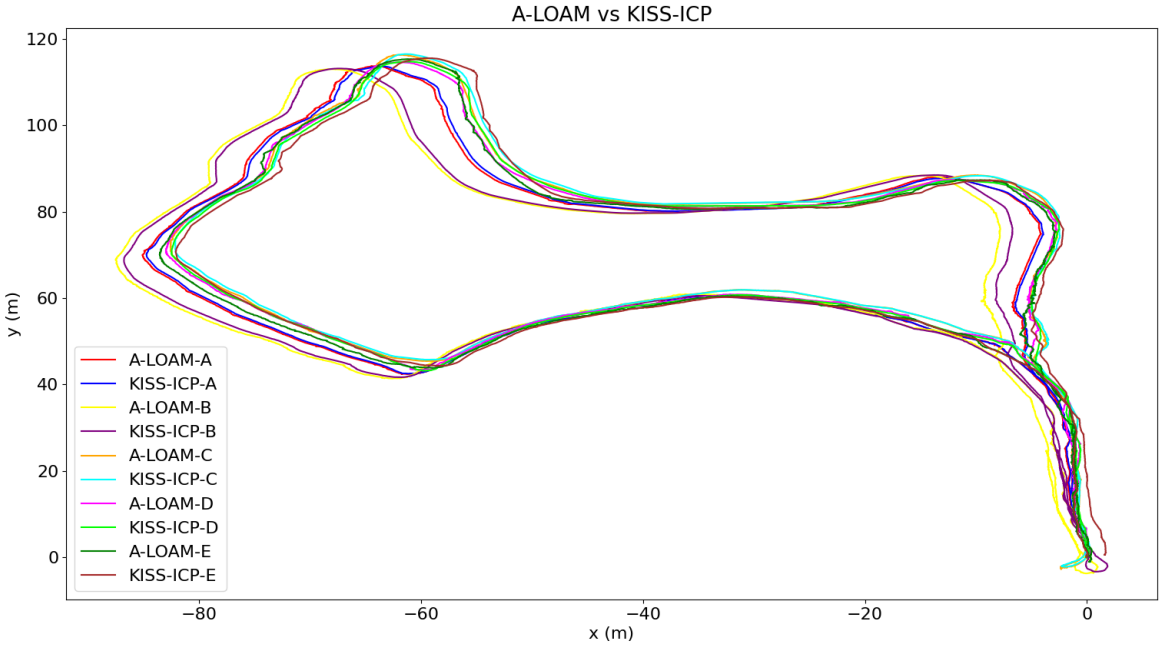

Figure 5.52: Trajectories generated by A-LOAM and KISS-ICP.

Figure 5.53: 3D map generated by A-LOAM.

A 3D map of one of the runs built from A-LOAM is shown in 5.53. The map is built using one scan for every 5 scans and each scan's range is restricted to 8m to build a more refined map. A-LOAM works very well in outdoor environments as the "loops" are nearly closed(i.e., low drift) in all the experiments as shown in Figure 5.52. This is due to the slow speed of the Husky, which does not cause enough rotation to create distortion.

KISS-ICP also demonstrated good performance on this dataset. However, certain configuration adjustments were necessary to optimise its accuracy in the outdoor environment. Specifically, the voxel size of the map was increased to 0.5 metres, and the minimum LiDAR range was set to 2.0 metres.

5.8 Summary

The experimental results across various datasets and algorithms are summarized in Table 5.5. Overall, FAST-LIO2 demonstrated robust performance and low drift in many scenarios. However, on datasets with fast rotations and missing point timestamps, such as HKUST, FAST-LIO2 encountered challenges and was outperformed by A-LOAM. KISS-ICP, while not tested on the HKUST dataset due to its similarity to A-LOAM, generally showed small drifts in less aggressive environments but suffered from significant drift under more challenging conditions. Notably, on the UCT Walking dataset, FAST-LIO2 proved to be robust under aggressive motion, while KISS-ICP exhibited varying degrees of drift depending on the scenario. These results highlight the importance of precise motion compensation and the limitations of LiDAR-only odometry in highly dynamic environments.

Table 5.5: Summary of Experimental Results

Experiment/Dataset	**Algorithm**	**Performance**
HKUST	A-LOAM	Outperformed FAST-LIO2.
HKUST	FAST-LIO2	Failed in fast rotations due to lack of point timestamps.
Rotation	KISS-ICP	Significant drift observed.
Rotation	FAST-LIO2	Performed well; low drift relative to ground truth (LIO-SAM).
NCD	KISS-ICP	Small drift observed.
NCD	FAST-LIO2	Outperformed KISS-ICP in all experiments.
UCT Rotation	KISS-ICP	Significant drift observed.
UCT Rotation	FAST-LIO2	Small drift observed.
UCT Walking	KISS-ICP	Significant drift observed.
UCT Walking	FAST-LIO2	Small drift. Robust under aggressive motion.
UCT Zoo	A-LOAM	Little to no drift. Similar performance as KISS-ICP.
UCT Zoo	KISS-ICP	Little to no drift. Similar performance as A-LOAM.

Chapter 6

Discussions and Conclusions

This thesis has presented an advanced LiDAR odometry and mapping pipeline, FAST-LIO2, designed for challenging conditions that involve high linear and angular acceleration. The primary objective was to produce precise 3D maps in scenarios where traditional methods might struggle due to aggressive motion.

6.1 Key Findings

Through extensive evaluation using the UCT Zoo dataset and additional datasets from [23] and [43], FAST-LIO2 demonstrated robust performance in various challenging environments. The results indicated that while deskewing is not always critical in slower, rugged terrain scenarios, it becomes indispensable under extreme motion conditions due to the relative speed of the robot compared to the LiDAR scanning rate. This highlights the importance of accurate point timestamps in maintaining the integrity of LiDAR data when subjected to rapid movement.

6.2 Comparison with KISS-ICP

In a comparative evaluation with FAST-LIO2, it was observed that KISS-ICP performed effectively in most scenarios but showed limitations in handling yaw motion. KISS-ICP, which utilises a constant velocity model, was found to be less accurate in yaw-heavy experiments. This failure underscores the critical role of precise motion modelling in LiDAR odometry. The use of a constant velocity model in KISS-ICP likely resulted in inaccurate poses under yaw conditions, highlighting the need for robust motion compensation strategies.

6.3 Importance of an IMU

The findings of this thesis emphasise the importance of integrating an IMU into the LiDAR odometry system. An IMU provides crucial information about the vehicle's angular velocity and linear acceleration along three dimensions, which is essential for accurate motion compensation. This is particularly important for scenarios involving complex motions, including aggressive yaw changes, where accurate orientation information is necessary to prevent the types of failures observed with FAST-LIO2 in comparison to KISS-ICP.

6.4 Ablation Studies: Feature-based Matching vs. Point Matching

Additional ablation studies were conducted to compare feature-based matching and point matching. The results of these studies indicated that feature-based matching outperforms point matching in an indoor environment. Feature-based matching provided more accurate and reliable correspondences, leading to improved overall system performance. This finding suggests that focusing on feature extraction and matching can enhance the robustness and precision of LiDAR odometry, particularly in challenging scenarios involving rapid motion.

Overall, this thesis confirms that FAST-LIO2 is a robust registration algorithm for creating accurate 3D maps in environments characterized by aggressive motion. The comparative analysis with KISS-ICP highlighted areas for improvement, particularly in handling yaw motions. The findings underscore the importance of incorporating robust motion models and leveraging 6-axis IMU data to enhance the accuracy of motion estimation. By addressing these aspects, future iterations of the system can achieve even greater robustness across a wide range of scenarios.

Chapter 7

Further Work

While FAST-LIO2 has demonstrated its effectiveness in scenarios involving aggressive motion, there are several areas where improvements can be made. This section outlines potential future work to address current limitations and propose possible solutions.

7.1 Experiments to better understand aggressive motion

To better comprehend the complexities of aggressive motion involving LiDAR, further experiments need to be conducted. The datasets should encompass a wide range of aggressive motion patterns, including rapid accelerations, abrupt turns, and rotations. This diversity ensures comprehensive coverage of potential real-world scenarios. Additionally, LiDAR data should be collected in various environments, including urban, rural, and indoor settings, to capture the effects of different surroundings on LiDAR performance under aggressive motions. Different types of LiDAR sensors should be utilised to evaluate the algorithm under varying conditions. In addition to this, incorporating multiple sensors in these datasets will provide more accurate ground truth data.

7.2 Deep learning models

Deep learning approaches aim to address the correspondence matching issue inherent in ICP-based approaches. Traditional ICP algorithms rely on finding the closest point pairs between two point clouds, which can be computationally intensive and prone to converging to a local minimum. However, recent advancements in deep learning have led to more robust and efficient correspondence matching. Further research should explore the robustness of deep learning methods such as LO-Net [59], PointNet [60] and PointNet++ [61] in scenarios involving aggressive motions.

7.3 Sensor-Agnostic Feature Extraction and Loop Closure

An important area for improvement is the development of a sensor-agnostic feature extraction process and the integration of a loop closure mechanism. Although there are some implementations of FAST-LIO2 with loop closure, they are limited in the types of sensors used. Designing

a feature extraction process that works independently of the LiDAR sensor type will make the system more versatile. The addition of loop closure will reduce drift and enhance map accuracy, making the system more reliable for long-term deployments.

By addressing these areas, FAST-LIO2 can be significantly enhanced to handle challenging scenarios, diverse environments and different sensors. This will not only improve the system's performance but also expand its usability to more challenging and diverse real-world scenarios.

Bibliography

[1] Milad Ramezani, Yiduo Wang, Marco Camurri, David Wisth, Matias Mattamala, and Maurice Fallon. The Newer College Dataset: Handheld LiDAR, inertial and vision with ground truth. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4353–4360, 2020.

[2] Waymo. Waymo vehicle image. Accessed: 2024-11-11.

[3] Zenmuse. Zenmuse LiDAR. Accessed: 2024-11-11.

[4] Lin Yang, Hongwei Ma, Zhen Nie, Heng Zhang, Zhongyang Wang, and Chuanwei Wang. 3D LiDAR point cloud registration based on IMU preintegration in coal mine roadways. *Sensors*, 23:3473, 03 2023.

[5] Wei Xu, Yixi Cai, Dongjiao He, Jiarong Lin, and Fu Zhang. FAST-LIO2: fast direct LiDAR-inertial odometry. *CoRR*, abs/2107.06829, 2021.

[6] Ignacio Vizzo, Tiziano Guadagnino, Benedikt Mersch, Louis Wiesmann, Jens Behley, and Cyrill Stachniss. KISS-ICP: In defense of point-to-point ICP – simple, accurate, and robust registration if done the right way. 2022.

[7] Ji Zhang and Sanjiv Singh. LOAM: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems X*. Robotics: Science and Systems Foundation, July 2014.

[8] P.J. Besl and Neil D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, February 1992.

[9] Y. Chen and G. Medioni. Object modeling by registration of multiple range images. In *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, pages 2724–2729 vol.3, 1991.

[10] Aleksandr Segal, Dirk Hähnel, and Sebastian Thrun. Generalized-ICP. 06 2009.

[11] Philippe Babin, Philippe Giguere, and Francois Pomerleau. Analysis of robust functions for registration algorithms. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, May 2019.

[12] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.

[13] Ji Zhang and Sanjiv Singh. Low-drift and real-time lidar odometry and mapping. *Autonomous Robots*, 41:401–416, 02 2016.

[14] HKUST Aerial Robotics. A-LOAM: Advanced implementation of LOAM. `https://github.com/HKUST-Aerial-Robotics/A-LOAM`, 2019. Accessed: 2025-06-29.

[15] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152, Quebec City, Que., Canada, 2001. IEEE Comput. Soc.

[16] Kenji Koide, Masashi Yokozuka, Shuji Oishi, and Atsuhiko Banno. Voxelized GICP for fast and accurate 3D point cloud registration. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11054–11059, 2021.

[17] Peter Biber and Wolfgang Straßer. The normal distributions transform: A new approach to laser scan matching. volume 3, pages 2743 – 2748 vol.3, 11 2003.

[18] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[19] Kenny Chen, Brett T. Lopez, Ali akbar Agha-mohammadi, and Ankur Mehta. Direct LiDAR odometry: Fast localization with dense point clouds, 2022.

[20] Jose Luis Blanco and Pranjal Kumar Rai. nanoflann: a C++ header-only fork of FLANN, a library for nearest neighbor (NN) with KD-trees. `https://github.com/jlblancoc/nanoflann`, 2014.

[21] Erik Nelson and contributors. BLAM: Bundle adjustment in the large. `https://github.com/erik-nelson/blam`, 2024.

[22] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2D LIDAR SLAM. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1271–1278. IEEE, 2016.

[23] Haoyang Ye, Yuying Chen, and Ming Liu. Tightly coupled 3D lidar inertial odometry and mapping. pages 3144–3150, 05 2019.

[24] Matteo Palieri, Benjamin Morrell, Abhishek Thakur, Kamak Ebadi, Jeremy Nash, Arghya Chatterjee, Christoforos Kanellakis, Luca Carlone, Cataldo Guaragnella, and Ali-akbar Agha-Mohammadi. Locus: A multi-sensor lidar-centric solution for high-precision odometry and 3D mapping in real-time. *IEEE Robotics and Automation Letters*, 6(2):421–428, 2020.

[25] Bo Zhou, Yi He, Kun Qian, Xudong Ma, and Xiaomao Li. S4-slam: A real-time 3D LIDAR SLAM system for ground/watersurface multi-scene outdoor applications. *Autonomous Robots*, 45, 01 2021.

[26] Nicolas Mellado, Dror Aiger, and Niloy J. Mitra. Super 4PCS fast global pointcloud registration via smart indexing. *Computer Graphics Forum*, 33(5):205–215, 2014.

[27] Richard A Newcombe, Andrew Fitzgibbon, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, and Steve Hodges. KinectFusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*. IEEE, October 2011.

[28] Frank Neuhaus, Tilman Koß, Robert Kohnen, and Dietrich Paulus. *MC2SLAM: Real-time inertial lidar odometry using two-scan motion compensation: 40th German Conference, GCPR 2018, Stuttgart, Germany, October 9-12, 2018, Proceedings*, pages 60–72. 01 2019.

[29] Pierre Dellenbach, Jean-Emmanuel Deschaud, Bastien Jacquet, and Francois Goulette. What's in my LiDAR odometry toolbox? In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, September 2021.

[30] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. pages 3354–3361, 05 2012.

[31] Gaurav Pandey, James McBride, and Ryan Eustice. Ford campus vision and lidar data set. *I. J. Robotic Res.*, 30:1543–1552, 10 2011.

[32] Nicholas Carlevaris-Bianco, Arash K. Ushani, and Ryan M. Eustice. University of Michigan North Campus long-term vision and lidar dataset. *International Journal of Robotics Research*, 35(9):1023–1035, 2015.

[33] Pierre Dellenbach, Jean-Emmanuel Deschaud, Bastien Jacquet, and François Goulette. CT-ICP: Real-time elastic LiDAR odometry with loop closure. September 2021.

[34] Jean-Emmanuel Deschaud. IMLS-SLAM: Scan-to-model matching based on 3D data. 02 2018.

[35] Yue Pan, Pengchuan Xiao, Yujie He, Zhenlei Shao, and Zesong Li. MULLS: Versatile LiDAR SLAM via multi-metric linear least square. pages 11633–11640, 05 2021.

[36] Han Wang, Chen Wang, Chun-Lin Chen, and Lihua Xie. F-LOAM: Fast LiDAR odometry and mapping. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, September 2021.

[37] Jinyong Jeong, Younggun Cho, Young-Sik Shin, Hyunchul Roh, and Ayoung Kim. Complex urban LiDAR data set, 2018.

[38] Frank Dellaert and Michael Kaess. *Factor graphs for robot perception.* Foundations and Trends in Robotics, Vol. 6, 2017.

[39] Masashi Yokozuka, Kenji Koide, Shuji Oishi, and Atsuhiko Banno. LiTAMIN: LiDAR-based tracking and mapping by stabilized ICP for geometry approximation with normal distributions. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5143–5150, 2020.

[40] Masashi Yokozuka, Kenji Koide, Shuji Oishi, and Atsuhiko Banno. Litamin2: Ultra light LiDAR-based SLAM using geometric approximation applied with KL-divergence, 2021.

[41] Heng Yang, Jingnan Shi, and Luca Carlone. TEASER: Fast and certifiable point cloud registration. *IEEE Transactions on Robotics*, 37(2):314–333, 2021.

[42] Tong Qin, Peiliang Li, and Shaojie Shen. VINS-Mono: A robust and versatile monocular visual-inertial state estimator. *CoRR*, abs/1708.03852, 2017.

[43] Tixiao Shan, Brendan Englot, Drew Meyers, Wei Wang, Carlo Ratti, and Daniela Rus. LIO-SAM: Tightly-coupled lidar inertial odometry via smoothing and mapping. pages 5135–5142, 10 2020.

[44] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. On-manifold preintegration for real-time visual–inertial odometry. *IEEE Trans. Robot.*, 33(1):1–21, February 2017.

[45] Tixiao Shan and Brendan Englot. LeGO-LOAM: Lightweight and ground-optimized lidar odometry and mapping on variable terrain. pages 4758–4765, 10 2018.

[46] Kenny Chen, Ryan Nemiroff, and Brett T. Lopez. Direct LiDAR-inertial odometry: Lightweight LIO with continuous-time motion correction, 2023.

[47] Lintong Zhang, Marco Camurri, David Wisth, and Maurice Fallon. Multi-camera LiDAR inertial extension to the Newer College Dataset, 2021.

[48] Chao Qin, Haoyang Ye, Christian Pranata, Jun Han, Shuyang Zhang, and Ming Liu. LINS: A lidar-inertial state estimator for robust and efficient navigation. pages 8899–8906, 05 2020.

[49] Zheng Huai and Guoquan Huang. Robocentric visual-inertial odometry. *CoRR*, abs/1805.04031, 2018.

[50] Wei Xu and Fu Zhang. FAST-LIO: A fast, robust lidar-inertial odometry package by tightly-coupled iterated Kalman filter. *CoRR*, abs/2010.08196, 2020.

[51] Christoph Hertzberg, René Wagner, Udo Frese, and Lutz Schröder. Integrating generic sensor fusion algorithms with sound state representations through encapsulation of manifolds. *CoRR*, abs/1107.1119, 2011.

[52] Yixi Cai, Wei Xu, and Fu Zhang. ikd-tree: An incremental kd tree for robotic applications. *arXiv preprint arXiv:2102.10808*, 2021.

[53] Dongjiao He, Wei Xu, and Fu Zhang. Kalman filters on differentiable manifolds. *arXiv preprint arXiv:2102.03804*, 2021.

[54] Dongjiao He, Wei Xu, Nan Chen, Fanze Kong, Chongjian Yuan, and Fu Zhang. Point-LIO: Robust high-bandwidth light detection and ranging inertial odometry. *Advanced Intelligent Systems*, 5, 07 2023.

[55] Fangcheng Zhu, Yunfan Ren, and Fu Zhang. Robust real-time LiDAR-inertial initialization, 2022.

[56] François Pomerleau, Francis Colas, Roland Siegwart, and Stéphane Magnenat. Comparing ICP variants on real-world data sets. *Autonomous Robots*, 04 2013.

[57] Krishtof Korda. pcap-to-bag. `https://github.com/Krishtof-Korda/pcap-to-bag`, 2020.

[58] Ouster Lidar. ouster_example. `https://github.com/ouster-lidar/ouster_example/releases/tag/20210608`, 2021.

[59] Qing Li, Shaoyang Chen, Cheng Wang, Xin Li, Chenglu Wen, Ming Cheng, and Jonathan Li. LO-Net: Deep real-time lidar odometry, 2020.

[60] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation, 2017.

[61] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space, 2017.