

Position Estimation of Intelligent Artificial Systems Using 3D Point Cloud

by

VRAJ PATEL
202111016

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of

MASTER OF TECHNOLOGY
in
INFORMATION AND COMMUNICATION TECHNOLOGY
to

DHIRUBHAI AMBANI INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGY



July, 2023

Declaration

I hereby declare that

- i) the thesis comprises of my original work towards the degree of Master of Technology in Information and Communication Technology at Dhirubhai Ambani Institute of Information and Communication Technology and has not been submitted elsewhere for a degree,
- ii) due acknowledgment has been made in the text to all the reference material used.



VRAJ PATEL

Certificate

This is to certify that the thesis work entitled POSITION ESTIMATION OF INTELLIGENT ARTIFICIAL SYSTEMS USING 3D POINT CLOUD has been carried out by VRAJ PATEL for the degree of Master of Technology in Information and Communication Technology at *Dhirubhai Ambani Institute of Information and Communication Technology* under my supervision.



PROF. TAPAS KUMAR MAITI

Thesis Supervisor

Acknowledgments

I would like to express my heartfelt gratitude to my supervisor, Prof. Tapas Kumar Maiti, whose expertise and unwavering commitment to academic excellence have guided me throughout this thesis. His insightful feedback, constructive criticism, and constant motivation have been instrumental in shaping the direction and quality of my research. I am truly grateful for his guidance and the opportunities they have provided me to grow as a researcher.

I would also like to extend my sincere thanks to Mr. Aditya Bhoje, a fellow researcher, for his invaluable support. His willingness to lend a helping hand and his contributions have enriched my understanding and significantly improved the outcomes of this research. I am grateful for his intellectual synergy and the camaraderie I have developed throughout this journey.

Furthermore, I would like to express my deep appreciation to my dear friend, Harsh. His unwavering belief in my abilities, encouragement during challenging times, and willingness to provide a fresh perspective have been a constant source of inspiration. His support and friendship have played a crucial role in keeping me motivated and reminding me of the importance of balance and well-being during this demanding process. I am truly grateful for his presence in my life and the impact he had on my personal and academic growth.

Contents

Abstract	vi
List of Principal Symbols and Acronyms	vi
List of Figures	vii
1 Introduction	1
1.1 Introduction to Point Cloud	1
1.2 Applications of Point Cloud	2
1.3 Challenges in Working with Point Cloud	4
1.4 Point Cloud for Intelligent Artificial Systems	5
1.5 Literature Review	6
2 Robot Operating System	10
2.1 Introduction	10
2.2 ROS Communication	11
2.3 Libraries and Tools	13
2.3.1 Gazebo Simulator	13
2.3.2 Point Cloud Library	14
2.3.3 Ceres Solver	16
2.3.4 OctoMap	17
2.3.5 OpenCV	18
2.3.6 ROS Bag	18
2.3.7 Rviz	18
2.3.8 Catkin	19
2.3.9 ROS bash	19
2.3.10 ROS launch	19
3 Edge Device	20
3.1 Raspberry Pi	20
3.1.1 Introduction	20

3.1.2	Components of Raspberry Pi	20
3.2	Intel Realsense LiDAR Camera L515	22
3.2.1	Introduction	22
3.2.2	Specifications	22
3.2.3	Stereo-LiDAR Technology	22
4	Position Estimation	23
4.1	Simultaneous Localization and Mapping (SLAM)	23
4.1.1	Types of SLAM	26
4.1.2	Challenges of SLAM	30
4.2	Methodology	32
4.2.1	Feature Extraction	32
4.2.2	Odometry Estimation	33
4.2.3	Probablity Map Construction	34
4.2.4	Novelty of the Proposed Methodology	35
5	Experimental Verification	36
5.1	Experimental Setup	36
5.2	3D Point Cloud Dataset Prepared Using LiDAR	38
5.3	Result and Discussion	41
6	Conclusions	50
	References	53

Abstract

The three-dimensional reality collected by various sensors such as LiDAR scanners, depth cameras and stereo cameras, is represented by point cloud data. The capacity of point clouds to provide rich geometric information about the surroundings makes them essential in various applications. Robotics, autonomous cars, augmented reality, virtual reality, and 3D reconstruction all use point clouds. They allow for object detection, localization, mapping, scene comprehension, and immersive visualization. Working with point clouds, on the other hand, presents substantial complications. Some primary issues are managing a vast volume of data, dealing with noise and outliers, dealing with occlusions and missing data, and conducting efficient processing and analysis. Furthermore, point clouds frequently necessitate complicated registration, segmentation, feature extraction, and interpretation methods, necessitating computationally costly processing. Addressing these issues is critical for realizing the full potential of point cloud data in a variety of real-world applications.

SLAM is a key technique in robotics and computer vision that addresses the challenge of estimating a robot's pose and constructing a map of its environment. It finds applications in driverless cars, drones, and augmented reality, enabling autonomous navigation without external infrastructure or GPS. Challenges include sensor noise, drift, and uncertainty, requiring robust sensor calibration, motion modeling, and data association. Real-time speed, computing constraints, and memory limitations are important considerations. Advanced techniques such as feature extraction, point cloud registration, loop closure detection, and Graph-SLAM optimization algorithms are used. Sensor fusion, map representation, and data association techniques are vital for reliable SLAM performance.

The aim is to create a compact and lightweight LiDAR based SLAM that can be easily integrated into various platforms without compromising on the accuracy and reliability of SLAM algorithms. Hence, we implemented a lightweight SLAM algorithm on our dataset with various background situations and a few modifications to the existing SLAM algorithm to improve the results. We have performed SLAM by using LiDAR sensor without the use of IMU or GPS sensor. The

lightweight LiDAR SLAM has significant implications for various fields, including robotics, autonomous navigation, and augmented reality. Developing compact and efficient LiDAR SLAM systems makes it possible to unlock the potential of lightweight platforms, enabling their deployment in a wide range of applications that require real-time position mapping, and localization capabilities while ensuring practicality, portability, and cost-effectiveness.

List of Figures

1.1	Demo Point cloud	1
1.2	Applications of Point Clouds	3
2.1	ROS Architecture	11
3.1	Raspberry Pi Components	21
4.1	Method Overview	32
5.1	Physical Experimental Setup	37
5.2	Experimental Setup	39
5.3	Dataset Description	40
5.4	Results obtained with existing algorithm on demo dataset [9]	42
5.5	Results obtained with our improved algorithm on demo dataset	42
5.6	[9]’s Code on our bright dataset	43
5.7	Our Code on our bright dataset	44
5.8	[9]’s Code on our dark dataset	45
5.9	Our Code on our dark dataset	47
5.10	[9]’s Code on our completely dark dataset	48
5.11	Our Code on our completely dark dataset	48
5.12	Estimated Odometry vs Actual Odometry	49

CHAPTER 1

Introduction

1.1 Introduction to Point Cloud

Point clouds are 3D representations of a collection of points in space as depicted in Fig. 1.1. Each point in a PointCloud is represented by its position in 3D space, which is described by the coordinates (x, y, z) relative to a given reference frame. Furthermore, each point may have related attributes such as color, intensity, or reflectivity, which can provide extra information on the physical properties of the surroundings. A Point cloud can be described mathematically as a set of points in 3D space, each defined by a vector:

$$p = [x, y, z, r, g, b, \dots] \quad (1.1)$$

Where $x, y,$ and z are the point's 3D coordinates, and $r, g,$ and b are the color values (red, green, and blue, respectively) associated with the point. This depiction can also incorporate other qualities, such as intensity or reflectivity.

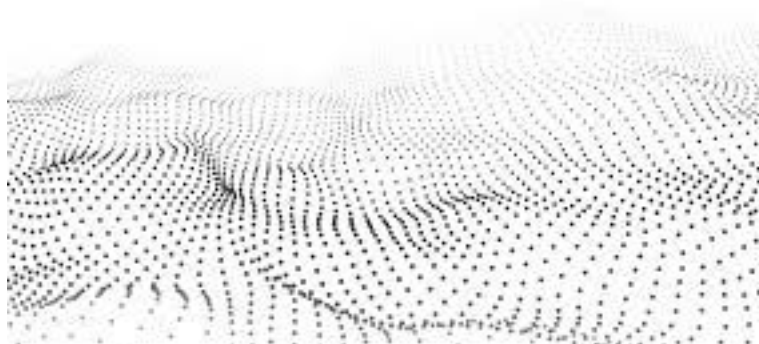


Figure 1.1: Demo point cloud [1]

Image courtesy: <https://www.sigarch.org/point-clouds-are-eating-the-world/>

PointClouds are often captured using sensors such as LiDAR or depth cameras, which emit a beam of light to measure the time it takes to reflect off an object in the environment and return to the sensor. This is known as time-of-flight measurement, enabling the sensor to estimate the distance between itself and the item. The sensor may build a 3D picture of the environment as a PointCloud by integrating information from several beams fired at different angles. The density of PointClouds refers to the number of points per unit area or volume, is an important factor. PointClouds with a higher density include more points, providing a more detailed depiction of the surroundings. Higher-density PointClouds, on the other hand, necessitate more storage and processing power and is often more challenging to work with.

To extract relevant information from point cloud data, a variety of techniques, such as point cloud registration, segmentation, or grouping, can be used. Point cloud registration, for example, entails aligning two or more PointClouds that record the same environment from multiple perspectives to form a unified, coherent representation of the environment. This is extensively utilized in 3D scanning, robotics, and virtual reality applications.

In summary, PointClouds are 3D representations of a collection of points in space often generated by LiDAR or depth cameras. They are theoretically represented as vectors, each corresponding to a point in 3D space and perhaps including other qualities such as color or reflectivity. PointClouds have a plethora of uses and are processed using various algorithms to extract useful information from the data.

1.2 Applications of Point Cloud

PointClouds have numerous uses in a variety of areas as illustrated in Fig. 1.2. Because of their ability to record and represent comprehensive 3D spatial information, they are a valuable tool for activities such as perception, modeling, and analysis. Their use is likely to increase as more sectors recognize their potential. Because of its ability to gather and represent precise 3D spatial information, point clouds have extensive applications in a wide range of sectors. Here are a few examples of how PointClouds are used in different fields:

1. **Robotics and Driverless Vehicles:** PointClouds are used to sense and navigate the surroundings in robotics and driverless vehicles. Robots and automated vehicles can recognize obstacles, determine paths, and prevent colli-

sions by analyzing PointCloud data. For example, self-driving cars generate PointClouds of their surroundings using LiDAR sensors, which are subsequently processed to recognize road signs, traffic signals, and other vehicles.

2. **3D Modelling and Scanning:** PointClouds are frequently used to build detailed digital representations of physical objects or settings in 3D modeling and scanning applications. Architects and designers, for example, can utilize PointClouds to scan existing buildings or landscapes and create precise digital models for planning and design. PointClouds can also construct realistic virtual settings for video games, movies, or VR experiences.
3. **Augmented Reality and Mixed Reality:** PointClouds give a more immersive and realistic experience in augmented reality and mixed reality applications. Augmented reality apps can generate a more convincing appearance of interaction between the real and virtual worlds by superimposing virtual objects on a PointCloud of the real environment. An augmented reality program, for example, may use a PointCloud of a room to insert virtual furniture or other objects in the real world.



Figure 1.2: Applications of point clouds: robotics and driverless vehicles, augmented reality, archaeological and cultural heritage, and 3D modelling [This work] [2, 3, 4]

Adapted from: <https://lidarnews.com/articles/the-zamani-project-heritage-documentation-beyond-the-point-cloud/>

<https://www.bisinfotech.com/top-technologies-shaping-metaverse>

<https://mobilitybehaviour.eu/2017/08/07/perspectives-on-the-future-of-the-car/>

4. **Archaeology and Cultural Heritage:** Archaeological and cultural heritage sites can be captured and preserved using PointClouds. Researchers and historians can better comprehend the layout and construction of these places by creating precise 3D models, allowing them to analyze them in greater depth than standard 2D images or sketches. For example, a PointCloud of a historic structure or landmark may create an interactive digital tour or educational experience.
5. **Environmental Monitoring:** PointClouds track and analyze environmental changes over time. PointClouds created by satellite photography or drone surveys, for example, can be used to track changes in vegetation, water levels, or land usage over time. This information can be used to predict natural disasters, evaluate ecosystem health, and detect regions of possible environmental damage.

1.3 Challenges in Working with Point Cloud

Working with 3D PointClouds is difficult due to their colossal size, intricacy, and lack of organization. The most significant issues associated with dealing with 3D PointClouds are as follows:

1. **Data Size and Storage[5]:** 3D PointClouds can be massive, with millions or even billions of points. This can make data storage and processing difficult, especially with real-time applications like robotics or autonomous cars. Processing activities such as registration, segmentation, or classification may also necessitate extensive computational resources.
2. **Lack of Structure:** PointClouds lack inherent structure and can be constructed in any form. This can make analyzing or extracting relevant information from data challenging. To make sense of PointCloud data, algorithms such as segmentation or clustering are frequently used to group points together based on their qualities. Machine learning techniques are used to detect patterns or links in the data.
3. **Noise and Incomplete Data:** Due to sensor limits, ambient conditions, or occlusions, PointCloud data can be noisy or incomplete. This can make extracting correct information from data challenging and may necessitate pre-processing techniques like filtering or smoothing to remove undesired noise or fill in missing data.

4. **Registration[6] and Alignment:** When working with several PointClouds taken from different sensors or views, it is frequently required to align and register the data to build a coherent picture of the environment. This can be difficult since it entails recognizing common features or landmarks in the data and understanding how they correspond across the many PointClouds.
5. **Visualization:** Visualising PointCloud data might be complex due to the enormous amount of points, making it difficult to display the data clearly and relevantly [7]. PointClouds can be visualized in various ways, such as displaying the points as spheres or discs or using surface reconstruction algorithms to produce a mesh representation of the data.

To summarise, complexity in working with 3D PointClouds high due to data size and storage, a lack of structure, noise and incomplete data, registration and alignment, and visualization. Overcoming these obstacles frequently necessitates the employment of specialized methods and approaches, as well as substantial processing resources. However, as 3D data becomes increasingly important in various businesses, attempts are being made to develop more efficient and effective methods for working with PointClouds.

1.4 Point Cloud for Intelligent Artificial Systems

3D point clouds are essential to many autonomous driving systems since they allow the car to observe and navigate its surroundings in real time. Here are a few examples of how PointClouds are utilized in autonomous driving:

1. **Object Detection and Tracking:** Detection and tracking of items in the environment: PointClouds are used to detect and track objects in the environment, such as other vehicles, pedestrians, or obstructions. The system can recognize possible dangers and take appropriate action to avoid collisions by analyzing the form and position of the points in the PointCloud.
2. **Localization and Mapping:** PointClouds may generate a detailed map of one's surroundings, which can be utilized for localization and navigation. The system can establish its position and orientation by comparing the current PointCloud to the map and plan a safe and efficient journey through the surroundings.
3. **Semantic Segmentation[8]:** PointClouds can be segmented into different regions based on their semantic meaning, such as road surfaces, sidewalks,

or buildings. This allows the system to better understand the structure of the environment and make more informed decisions about navigating through it.

4. **Obstacle Avoidance:** PointClouds are utilized in real-time to detect and avoid obstructions. The system can detect possible risks and avoid collisions by analyzing the distance and position of the points in the PointCloud.
5. **Sensor Fusion:** Data from additional sensors, such as cameras or radar, can be integrated with PointClouds to produce a more complete and accurate image of the surroundings. The system may overcome the limits of individual sensors by fusing input from several sensors and making more reliable judgments about how to move around the environment.

3D PointClouds are essential to autonomous driving systems because they allow the vehicle to perceive and comprehend its surroundings in real-time. Autonomous cars may navigate complex and dynamic settings with great accuracy and safety by using PointClouds for object recognition and tracking, localization and mapping, semantic segmentation, obstacle avoidance, and sensor fusion.

Traditional LiDAR sensors are often bulky, heavy, and power-hungry, making them unsuitable for many applications, such as small autonomous vehicles, drones, or wearable devices. The solid-state LiDAR has become popular since it provides a lightweight and cost-effective solution for small-scale robotics applications. Compared to mechanical LiDAR, solid-state LiDAR sensors have higher angular resolution and update frequency but also have a smaller field of view (FoV), which is very challenging for existing LiDAR SLAM algorithms. Hence, making it necessary to develop a computationally efficient, accurate, and robust SLAM algorithm for solid-state LiDAR. Taking inspiration from [9], we have developed a Lightweight SLAM to address this issue. We have performed several experiments on hand held device to prove the effectiveness of our method to eradicate the use of IMU sensors, along with validating the use of LiDAR-based lightweight SLAM in different lighting scenarios.

1.5 Literature Review

The literature review section provides a comprehensive overview of the existing research and developments in 3D Point cloud-based SLAM. It examines relevant studies, methodologies, and findings in order to establish the current state of

knowledge and identify gaps that the present research aims to address. By critically analyzing and synthesizing the existing literature, this review sets the stage for the subsequent sections of the thesis, highlighting the significance of the current research and the unique contribution it aims to make.

The Point cloud Library (PCL), an open-source library developed for 3D Point cloud processing, was introduced in [10]. It provides an overview of PCL's motivation and architecture. The authors cover a plethora of PCL functions, such as filtering, segmentation, registration, and feature extraction. They emphasize PCL's flexibility and modularity, allowing users to handle and analyze Point cloud data efficiently. The report also discusses numerous real-world applications where PCL has been used successfully.

In [11], the authors focus on visual simultaneous localization and mapping (SLAM) using RGB-D data, frequently represented as 3D point clouds. The authors thoroughly overview the various methods and strategies used in visual SLAM employing RGB-D sensors. They go over the fundamentals of visual SLAM, such as point cloud registration, loop closure detection, and map optimization. The study discusses several approaches and methodologies, emphasizing their advantages and disadvantages. It also identifies research problems and future directions in RGB-D SLAM.

The fast point feature histograms (FPFH) approach, which is extensively used for 3D Point cloud registration, is introduced in [12]. The authors describe a robust and efficient method for computing local feature descriptors based on point geometric parameters. FPFH gathers information about the local surface structure, allowing for precise and dependable 3D point cloud registration. The report includes experimental data and comparisons with alternative registration methods that demonstrate the efficacy of FPFH in a variety of settings.

The challenge of surface reconstruction from unorganized 3D point clouds is addressed in [13]. The authors give an insight into various methodologies and techniques for regenerating smooth surfaces from Point cloud data. They review techniques, including Delaunay triangulation, Poisson surface reconstruction, and implicit surface representations. The research investigates the benefits and drawbacks of each approach, shedding light on the trade-offs involved in surface reconstruction using point clouds. There are numerous examples of recreated surfaces.

The authors provide an in-depth examination of 3D local feature descriptors used in Point cloud analysis and matching. The authors look at various descriptors, such as spin pictures, 3D form contexts, and 3D histograms. Extensive tests

assess each descriptor’s robustness, discriminative strength, and processing efficiency [14]. The paper sheds light on the performance and applicability of numerous descriptors for diverse applications, assisting researchers and practitioners in picking acceptable descriptors for their needs.

The authors of aim to introduce Open3D, a new open-source toolkit built for 3D data processing, such as Point cloud manipulation, visualization, and reconstruction [15]. The authors give a high-level overview of the library’s functionality, emphasizing its efficient algorithms and user-friendly APIs. Open3D provides various functions, including filtering, registration, and surface reconstruction, making it an invaluable tool for researchers and practitioners working with 3D Point cloud data. The document contains examples of how to use Open3D for various 3D Point cloud processing applications.

In [16], the authors discuss effective SLAM employing 3D laser rangefinders, which generate 3D Point cloud data in the form of surfels (surface components). The authors offer a surfel-based SLAM approach with a compact surface representation and a hierarchical surface map structure. The approach enables the processing of large-scale settings in real-time while preserving accuracy. The research provides experimental evaluations and comparisons with previous SLAM methodologies, demonstrating the proposed method’s efficiency and effectiveness.

The authors of [17] provides an effective technique for estimating the motion of a LiDAR sensor in real time that combines the point-to-plane ICP (iterative closest point) algorithm with scan-to-scan odometry. The technique provides accurate registration between consecutive scans using the point-to-plane ICP approach, which aligns point clouds based on surface normals. The technique also uses scan-to-scan odometry to determine the sensor’s motion, allowing for accurate localization and mapping.

In [18], the authors describe a voxel-based mapping strategy for autonomous cars that incorporates real-time LiDAR odometry. The technique employs a volumetric representation, breaking the surroundings into voxel grids. It uses efficient map update algorithms to enable the real-time operation and to keep the map constantly updated with new sensor input. The technique enables accurate localization and mapping for autonomous navigation by combining LiDAR odometry, which estimates the vehicle’s velocity using LiDAR readings.

The authors of [19] offer a quick and robust 3-D mapping method utilizing a single scanning LiDAR sensor. The method prioritizes efficiency without sacrificing mapping accuracy. It uses adaptive voxel subsampling, which dynamically

adjusts voxel size based on local point density, decreasing computer complexity while preserving adequate map representation. Furthermore, the algorithm employs a scan-to-model registration technique, which makes use of a pre-built model to improve the registration process and provide accurate mapping results.

The authors of [9] describe a simple method for real-time 3-D localization and mapping with a solid-state LiDAR sensor. The approach combines a visual odometry system, which uses picture data to predict camera motion, with LiDAR odometry to provide precise and robust localization, particularly in harsh conditions. The technique increases the accuracy and reliability of the localization and mapping findings by combining information from both sensors.

In [20], the authors propose a series of efficient variations of the point-to-plane ICP algorithm for real-time LiDAR odometry. To reduce computational complexity while retaining accurate registration, the variants employ adaptive point sampling algorithms, which select points based on their saliency and distribution. Improved convergence criteria are also implemented to improve odometry estimates' convergence speed and accuracy.

These studies collectively contribute to lightweight 3-D localization and mapping for solid-state LiDAR and the application of 3D Point clouds. By addressing the challenges of computational efficiency while ensuring accurate mapping and localization results, these algorithms pave the way for the practical implementation of solid-state LiDAR technology in various applications, such as autonomous vehicles, robotics, and environmental mapping.

CHAPTER 2

Robot Operating System

2.1 Introduction

The Robot Operating System (ROS) is a free and open-source platform for creating and programming robots. ROS is a set of software frameworks and tools that assist developers in creating robot applications. Willow Garage created ROS in 2007, and the Open Robotics organization now maintains it.

This chapter provides a detailed overview of how ROS works and the pivotal libraries and components of ROS that are essential for the implementation of proposed system. The chapter begins by introducing ROS, followed by explaining the internal working of ROS and finally all the essential components and libraries used.

ROS is modular, with a wide variety of libraries and tools that can be used separately or in conjunction with other components. ROS is versatile because of its modularity, allowing developers to add new features to their robots. ROS has a distributed design, meaning that different components of a robot's software can operate on separate computers and communicate via a message system. This architecture enables greater flexibility in the design and construction of robots and better dependability and scalability. The ROS Core is a core component of ROS that provides the messaging infrastructure and other vital services to the rest of the system. ROS also contains a plethora of libraries for a variety of applications, such as robot navigation, sensor processing, and image recognition. Figure 2.1 depicts the ROS architecture. It comprises various components that work together to provide a distributed system for building and directing robots. The ROS-master is at the heart of the architecture. The ROS master handles communication between various aspects of the system, such as nodes, topics, services, and the parameter server. The 'roscore' command starts the ROS master, which must be running for other ROS components to communicate with each other.

The basic building elements of ROS applications are nodes. A node is a pro-

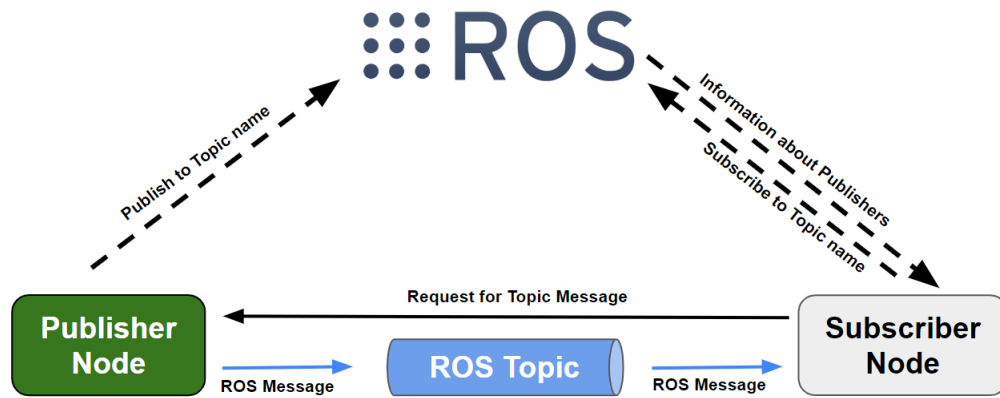


Figure 2.1: ROS architecture is illustrated.

cess that does something specialized, such as driving a robot’s motors or analyzing sensor data. Nodes communicate with one another using messages described in message files. Messages can be of simple data like integers, floats, and texts or more complicated data structures like arrays or custom data types. Topics are referred to as buses, through which nodes communicate. A topic may have multiple publishers and subscribers. Subscribers are nodes that receive messages from a topic, while publishers are nodes that post messages to a topic. Topics enable asynchronous data exchange between nodes. Services enable synchronous communication between nodes. A service is characterized by two messages: one for the request and another for the response. A client node sends a request message to a service server node, which processes the request and replies to the client node. ROS also contains a set of tools for visualizing and analyzing robot data. Rviz for example, is a 3D visualization application that allows users to view robot models, sensor data, and other data in 3D. RQt is a set of graphical user interface tools used to design and debug ROS applications.

2.2 ROS Communication

Nodes in ROS (Robot Operating System) connect via topics and services. In a distributed system, these communication channels allow nodes to share data and invoke remote functions. Let us take a closer look at each of these modes of communication.

1. ROS Topic

ROS topics allow for publish-subscribe communication. A node that wants to communicate data publishes it to a specific topic, which other interested nodes can subscribe to receive the data. Topics are commonly used for asyn-

chronous communication in which the sender and receiver nodes do not sync in time. Here is how topics are used to communicate:

A. Publishing Node: The publishing node is the node that publishes data on a topic. It generates a publisher object with the subject name, message type, and other parameters specified. The publisher object is in charge of sending data via the topic. The publishing node then uses the publisher object to generate messages of the given kind and publish them.

B. Subscribing Node: The subscribing node is a node that wishes to receive data from a subject. It generates a subscriber object with the specified topic name, message type, and callback method to handle the incoming data. The callback function is registered with the subscriber object by the subscribing node. The subscriber object calls the callback method whenever new data is published on the topic, sending the received message as an argument.

C. Message Types: Messages in ROS are defined using a message description language known as .msg files. These files specify a message's structure and data fields. Each message type has a distinct name, and nodes must agree on the message type to communicate properly.

D. Message Flow: Messages are sent to the topic by the publishing node and received by all subscribing nodes registered to that topic. Each message is delivered to each subscriber, who processes the data individually.

2. ROS Service

ROS services support synchronous request-response communication. A node can provide a service, and other nodes can request it. The node providing the service processes the request and responds to the requesting node. Services are often used for synchronous actions that require a response before progressing. Here, we discussed how services are used to communicate:

A. Service Provider Node: A service provider node is a node that provides a service. It defines a service by naming it and providing the request and response message types. In addition, the service provider node includes a callback function to receive requests and generate responses. The callback function is registered with the service by the service provider node.

B. Service Client Node: A service client node is a node that wants to request a service. It generates a client object with the service name, request message type, and response message type specified. The service client node then uses the client object to call the service, delivering the relevant request data. The call is held until the service provider responds.

C. Message Types: Service messages, like topics, are specified using .srv files

that explain the request and response message structure.

D. Communication Flow: A request message is sent by the service client node to the service provider node. The request is received, processed, and the service provider node generates a response message. The response is returned to the service client node, and the call is unblocked, allowing the client node to resume operation.

In summary, topics enable asynchronous publish-subscribe communication by allowing nodes to send and receive messages on specific topics. Services enable synchronous request-response communication, which allows nodes to send requests to services and get responses. These communication protocols serve as the foundation of ROS distributed design, allowing nodes in robotic systems to communicate and collaborate successfully.

2.3 Libraries and Tools

2.3.1 Gazebo Simulator

The Gazebo ROS package is a powerful tool for modeling complicated physical systems in a virtual environment in robotics and automation. It is made up of various parts that work together to create smooth integration between ROS and the Gazebo simulator.

The Gazebo simulator, responsible for constructing the virtual world, simulating the physics of the environment, and producing the visuals, is the primary component of the Gazebo ROS package. The simulator can simulate the mobility of robots and other types of sensors, such as cameras, LiDARs, and sonars. The ROS Gazebo Interface communicates between ROS and the Gazebo simulator. This interface connects ROS nodes to the Gazebo simulator, allowing users to control the simulation and retrieve sensor data.

Another component of the Gazebo ROS package is the ROS Control package, which offers the framework for controlling robots in Gazebo using ROS controllers. Before installing controllers on physical robots, users can build and test them in simulation. The Unified Robot Description Format (URDF) describes the robot model in the Gazebo simulator. The model defines the robot's physical properties, such as geometry, mass, and joints.

Finally, ROS nodes are discrete ROS system components that interface with the Gazebo simulator. They can operate the robot model in Gazebo by subscribing to topics, publishing messages, using services, and using actions. The Gazebo

ROS package, when combined, provides a wholesome simulation environment for testing and developing ROS-based robotic systems. Users may simulate complicated environments and test their robots under diverse conditions by combining Gazebo and ROS, allowing them to design more robust and reliable robots.

2.3.2 Point Cloud Library

The point cloud library (PCL) is a prominent open-source point cloud data processing library. It is incorporated into the Robot Operating System (ROS) environment, giving ROS users sophisticated point cloud processing capabilities. PCL offers a comprehensive range of point cloud processing algorithms, including filtering, segmentation, feature extraction, registration, and 3D reconstruction. These algorithms are intended to work with various sensors, including LiDAR, RGB-D, and stereo cameras.

PCL is used as a core library for point cloud processing in ROS. It includes a collection of ROS nodes that is used for a variety of point cloud processing tasks. These nodes are linked to form complicated point cloud processing pipelines. The key components of PCL Library are:

1. **Point Cloud Representation [21]:** Typically, point clouds are represented as a collection of points in a 3D Cartesian coordinate system. Each point is specified by its (x, y, z) coordinates and may or may not contain extra information like color, intensity, or surface normals.
2. **Data Structures:** PCL provides point cloud data structures such as `pcl::PointCloud` and `pcl::PointXYZ`, which contain point cloud data and individual point information.
3. **Filtering:** Filtering procedures in PCL remove point cloud noise, outliers, and irrelevant points. The voxel grid filter and the statistical outlier elimination filter are two extensively used filters.
 - (a) **Voxel Grid Filter:** The voxel grid filter separates the input point cloud into voxels of equal size and keeps just one representative point per voxel [22]. It aids in reducing point cloud density while keeping overall structure.
 - (b) **Statistical Outlier Removal Filter:** This filter finds outliers based on a statistical analysis of point neighborhoods [21]. The mean and standard deviation of the distances between each point and its neighbors

are computed. Outliers are points with distances greater than a defined criterion (mean + multiple * standard deviation).

4. **Segmentation:** In PCL, segmentation divides a point cloud into different sections based on characteristics such as planarity, curvature, or color. The RANSAC (Random Sample Consensus) algorithm is a common segmentation technique [23]. The RANSAC algorithm fits a model (for example, a plane) to a subset of randomly sampled points from the point cloud. It then computes the number of inliers, which are points within a given distance of the model. This method is repeated for a set number of iterations, and the model with the most inliers is deemed the best match.
5. **Registration [24]:** The goal is to align many point clouds into a single coordinate frame. PCL offers many registration strategies, including Iterative Closest Point (ICP) and Feature-based registration.

- (a) **ICP[25]:** Iterative Closest Point (ICP): The ICP algorithm minimizes the distance between matching points in two-point clouds iteratively. It begins with approximating the transformation between the source and target point clouds and refines iteratively. A 4x4 homogeneous matrix is commonly used to illustrate the transformation. The following objective function is minimized using the ICP algorithm:

$$\text{minimize } \sum ||p_i - T \cdot q_i||^2 \quad (2.1)$$

where p_i and q_i are corresponding points in the source and target point clouds, and T is the transformation matrix.

- (b) **Feature Extraction:** To create correspondences between different viewpoints, feature-based registration uses different features collected from point clouds. Keypoints, descriptors, and local surface characteristics are examples of these features. PCL includes feature extraction algorithms such as the SIFT (Scale-Invariant Feature Transform) and FPFH (Fast Point Feature Histograms) [12].
6. **Feature Extraction:** PCL provides feature extraction algorithms for calculating descriptors from point clouds that capture local geometry or appearance information. These descriptors can be used to perform tasks like object detection, registration, and scene reconstruction. Normal Estimation, Principal Component Analysis (PCA), and Local Reference Frames (SHOT) are examples of PCL descriptors.

7. **Visualization[26]** : PCL contains visualization tools for displaying point clouds and the outcomes of their processing. The visualization tool allows you to build windows, add point clouds, color code them, alter camera settings, and interactively examine the Point cloud data.
8. **Surface Reconstruction** : PCL includes techniques for regenerating surfaces from point clouds. One prevalent method is the Poisson surface reconstruction approach, which calculates a smooth surface based on the input Point cloud. It generates a waterproof surface mesh using a signed distance function and solving a Poisson equation.
9. **Octree Data Structure** : Octrees are hierarchical data structures that divide 3D space into smaller sections, making Point cloud processing and spatial querying more efficient. PCL provides an octree data structure for neighbor finding, voxelization, and spatial occupancy analysis.

2.3.3 Ceres Solver

Ceres Solver is a Google Research open-source C++ framework for tackling large-scale optimization problems [27]. It provides a solid foundation for modeling and solving nonlinear least squares, bundle adjustment, and generic nonlinear optimization problems [28]. It is widely used in various domains, such as computer vision, robotics, and scientific computing. It has an expressive and flexible syntax for defining optimization problems [27]. Users can construct cost functions, which indicate the goal to be minimized, as well as constraints, which define additional relationships or limitations on the variables. A nonlinear least squares problem is formed by combining these cost functions and restrictions.

$$\text{Cost Function} = \frac{1}{2} \left(f(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)^T \mathbf{W} f(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n) \right) \quad (2.2)$$

In this equation, the $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ represent the variables being optimized, and $f(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)$ is the vector-valued function that computes the residuals. The \mathbf{W} is an optional weighting matrix that assigns different weights to different residuals. The equation is multiplied by $\frac{1}{2}$ for convenience, as it does not affect the optimization process. Ceres solver employs a variety of numerical optimisation techniques to efficiently solve the defined issues. It is compatible with both local and global optimisation algorithms, such as resilient loss functions, trust region methods, and line search methods. To minimise the cost function, these methods iteratively update the variable values. The following steps demonstrate the

typical workflow of Ceres Solver:

1. **Define the Problem:** Define the variables, cost functions, and restrictions that will be used to solve the optimization problem. This entails defining the residual functions and assigning the variables' initial values.
2. **Configure the Solver:** Configure the solver parameters, such as the optimization algorithm, termination criteria, and optimization process control choices.
3. **Solve the Problem:** In order to minimize the cost function, use the solver to change the variables. Ceres Solver employs numerical optimization approaches to enhance variable values iteratively.
4. **Retrieve the Results:** Obtain the optimized variable values and other pertinent information, such as the ultimate cost value and convergence status.

2.3.4 OctoMap

OctoMap is a prominent open-source package used in robotics and computer vision for three-dimensional mapping [29]. It uses an octree data structure to give an efficient and compact representation of the environment. The octree divides space into cubic cells, allowing adaptive resolution and memory utilization. Here are some of the functionalities of OctoMap:

1. **Octree Data Structure:** In OctoMap, the environment is represented using the Octree data structure. It is a tree-like structure, each node representing a cubic cell in space. If the cell has a mixture of occupied and vacant space or is not sufficiently refined, the octree is recursively partitioned into eight child nodes, generating an octant. The root node represents the total map volume, whereas the leaf nodes hold occupancy information.
2. **Occupancy Probability:** OctoMap uses probability values to model the occupancy of each cell. The occupancy probability measures the chance of an obstacle occupying a cell. The occupancy probability is often updated and stored using a log-odds representation.
3. **Ray-Casting and Update Equations:** OctoMap updates the occupancy probability of cells based on sensor measurements using ray-casting. When a sensor measurement is received, a ray is thrown from the sensor's origin to the measurement's endpoint. The cells crossed by the ray are updated based on the measured value.

4. **Occupancy Threshold:** An occupancy threshold is used by OctoMap to assess whether a cell is occupied or free. Cells with occupancy probabilities greater than the threshold are considered occupied, whereas cells with probabilities less than the threshold are considered free.

2.3.5 OpenCV

OpenCV is a free and open-source machine learning and computer vision algorithm library [30]. This library contains methods that have already been implemented for computer vision applications. This library includes over 2000 optimized algorithms, including a mix of classic and state-of-the-art computer vision and machine learning techniques. ROS integrates with OpenCV, allowing us to leverage significant capabilities like object detection, tracking, and identification. ROS extends OpenCV with libraries like image pipelines that may be used for camera calibration, monocular image processing, stereo image processing, and depth image processing.

2.3.6 ROS Bag

ROS bag is a ROS file type used to record and play back messages on ROS topics. ROS bags save data from a ROS system to a file, which played back later for analysis, testing, or other uses.

A ROS bag file stores data in binary format, allowing for efficient storage and playback of enormous amounts of data. ROS bags can include messages from one or more topics, allowing data from various sensors or components to be recorded and played back in a single file.

Data logging and analysis are two of the most typical applications for ROS bags. For example, while doing a task, a robot may collect data from its sensors, which can then be saved to a ROS bag file. The data can then be analyzed to evaluate the robot's performance, discover flaws or issues, and improve the system's performance.

2.3.7 Rviz

The Robot Operating System (ROS) ecosystem includes the 3D visualization tool rviz. It allows you to visualize robot models, sensor data, and other information in a virtual environment, making debugging and understanding mechanical system behavior easier. Rviz enables users to see and interact with various forms of

data, such as point clouds, laser scans, pictures, and 3D models. Its versatile and adjustable interface allows users to arrange and configure the shown data to meet their specific requirements.

2.3.8 Catkin

The ROS structure is built using catkin. Based on CMake, Catkin is likewise cross-platform, open-source, and language-independent.

2.3.9 ROS bash

The utility to extend the capabilities of the bash shell is provided by the rosbash package. These tools, which replicate the functions of ls, cd, and cp, include rosls, roscd, and roscp. We can now use the ROS package name instead of the file path where the package is placed, thanks to recent updates to the ROS.

2.3.10 ROS launch

ROS launch is a Robot Operating System (ROS) method that allows you to start and configure numerous ROS nodes simultaneously. It allows you to design and manage the launch of multiple nodes in a single configuration file, set node settings, remap topic names, and specify node dependencies. The launch file is written in XML and is the main entry point for starting a ROS system. ROS launch streamlines starting and configuring numerous ROS nodes, making system configuration, modularity, and reusability easier. It offers a centralized method to manage the launch process and freedom in customizing node parameters, remapping topics, and dealing with node dependencies.

CHAPTER 3

Edge Device

This chapter intends to introduce the edge devices that have been used in this thesis. We have used two devices: (1) Raspberry Pi (2) LiDAR which are explained in detail in this chapter.

3.1 Raspberry Pi

3.1.1 Introduction

Raspberry Pi (see Fig. 3.1) is a lightweight single-board computer initially published in 2012 and then released several versions. It is divided into three distinct models: A, B, and Zero Raspberry pi. This device comprises a system-on-a-chip comprising an integrated CPU and GPU, onboard memory, and a 5V DC power unit. All Raspberry pi versions include a port for connecting with a dedicated camera and general-purpose Input/Output (GPIO) pins that can be used to connect with a variety of devices, such as LEDs, buttons, motors, power relays, and various types of sensors. In addition, some versions provide Ethernet and wireless communication options such as wifi and Bluetooth. Additionally, this machine has all of the capabilities of a standard computer. It also allows to connect the mouse, keyboard, and screen without any settings and an easy-to-use Linux desktop environment. It is not limited to one operating system; another OS can also be used in lightweight devices.

3.1.2 Components of Raspberry Pi

1. General Purpose Input/Output (GPIO): This is the Raspberry Pi's most important component. It connects various electronic components such as LED lights, motors, inductors, and relays. They read the electronic signal from the devices and also transfer it to the linked device.

2. Ethernet/USB/HDMI ports: The Ethernet connector allows to connect to the network through a cable. A USB connector is also available for connecting a mouse, keyboard, web camera, and USB devices. Moreover, the HDMI connector allows the screen to be shown on a projector or monitor.
3. Audio Jack: This component of the raspberry pi enables audio functioning. Connect headphones or a speaker to this component to make the system audible.
4. Camera Module Port: This port connects the Raspberry pi camera. Do not connect the web camera since it can connect to a USB port.
5. Micro USB power: Raspberry pi necessitates a 5V steady power supply. The power supply is therefore attached to this port.
6. Micro SD Card: Micro SD card is used to store the data. Here, the SD Card serves as the bootable card by storing the operating system and enabling the raspberry pi board to boot from it. It also functions as a hard drive, storing all the users' private files.

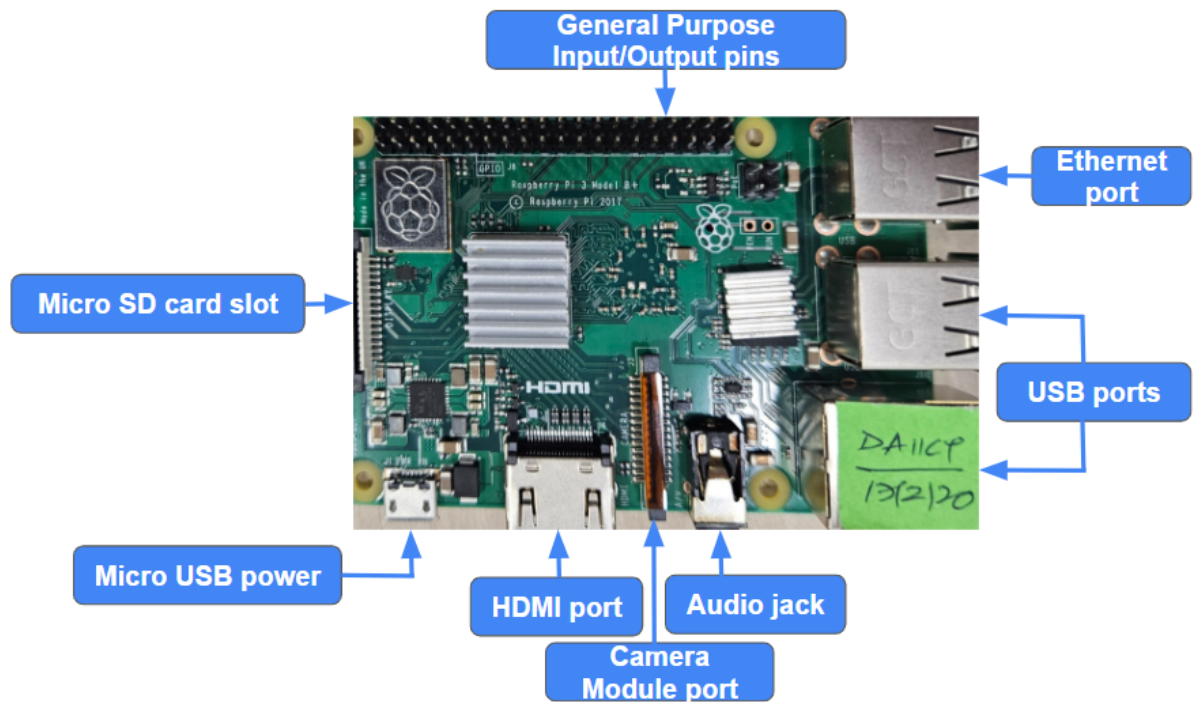


Figure 3.1: Illustrates the Raspberry Pi used in this work.

3.2 Intel Realsense LiDAR Camera L515

3.2.1 Introduction

The Intel RealSense L515 is a high-resolution LiDAR camera designed for various applications, including robotics, 3D scanning, and virtual reality. The L515 uses a unique "Stereo-LiDAR" technology to capture the depth and RGB data in real time.

3.2.2 Specifications

The Intel RealSense L515 has the following specifications:

- **Field of View (FoV):** 70° H x 55° V
- **Range:** Up to 9 meters
- **Resolution:** 1024 x 768
- **Frame Rate:** Up to 30 FPS
- **Depth Accuracy:** ±2cm (at 0.25-1m), ±3cm (at 1-2m), ±5cm (at 2-3m), ±7cm (at 3-4m), ±10cm (at 4-5m), ±15cm (at 5-6m), ±20cm (at 6-7m), ±30cm (at 7-8m), ±40cm (at 8-9m)

3.2.3 Stereo-LiDAR Technology

The Intel RealSense L515 employs a novel "Stereo-LiDAR" technique to gather depth and RGB data in real-time. A high-resolution RGB camera and a LiDAR sensor are used in this technique to create a very accurate and detailed Point cloud. The LiDAR sensor emits light via a laser beam, subsequently reflected by the environment. The time taken by the light to return is used to compute the distance between the item and the observer.

The RGB camera records color information for each point in the Point cloud, resulting in highly detailed 3D data that may be used for a variety of purposes: the L515's LiDAR sensor and RGB camera record depth and color data in real time.

CHAPTER 4

Position Estimation

This chapter discusses the backbone of this thesis SLAM. The chapter commences by introducing SLAM with appropriate mathematical expressions, followed by describing the types of SLAM with their pros and cons along with their typical use cases and finally the major challenges for SLAM have been described to conclude this chapter.

4.1 Simultaneous Localization and Mapping (SLAM)

Simultaneous Localization and Mapping is popularly known as SLAM. It is a fundamental challenge in robotics and computer vision that involves creating a map of an unfamiliar environment while determining the robot's position. Without pre-existing maps, SLAM algorithms allow autonomous systems to navigate and investigate their surroundings.

SLAM involves the robot collecting sensor measurements such as odometry (motion) data and observations from multiple sensors such as cameras, laser scanners, and depth sensors. These measurements are used to calculate the robot's pose (position and orientation) and to generate an environment map.

Consider a discrete-time formulation of SLAM, in which an intelligent system such as the robot's movements and sensor readings are recorded at discrete time steps. The following variables are defined:

1. **Robot Pose:** At time step t , the robot's pose can be represented by a vector x_t .

$$x_t = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \quad (4.1)$$

where (x, y) denotes the position coordinates, and θ represents the orientation.

2. **Robot Control:** The control input

$$u_t = \begin{bmatrix} u_x \\ u_y \\ u_\theta \end{bmatrix} \quad (4.2)$$

represents the robot's motion command between time steps $t - 1$ and t . It includes the translational displacement (u_x, u_y) and rotational displacement u_θ .

3. **Landmark Positions:** The environment can be represented by a set of landmarks, denoted by l_i

$$l_i = \begin{bmatrix} l_x^i \\ l_y^i \end{bmatrix} \quad (4.3)$$

Each landmark i has a 2D position (l_x^i, l_y^i) in the world frame.

4. **Robot Observation:** The robot's observations at time step t can be denoted by z_t^k , where k represents the k -th observation. Each observation consists of a measured range and bearing to a landmark.

Given the control inputs and observations, the core principle behind SLAM is maintaining a probability distribution across the robot's pose and landmark positions. The joint probability distribution commonly represents the posterior belief distribution:

$$P(x_t, l_1, \dots, l_N | z_1, \dots, z_t, u_1, \dots, u_t) \quad (4.4)$$

SLAM methods estimate this posterior using recursive filtering techniques such as the Particle Filter or the Extended Kalman Filter (EKF). Based on control inputs, these filters anticipate the robot's attitude (motion model) and update the belief using sensor data (measurement model).

The SLAM problem has two significant steps:

1. **Prediction (Motion Model):** The motion model predicts the robot's pose based on control inputs. It estimates the distribution $P(x_t | x_{t-1}, u_t)$.
2. **Update (Measurement Model):** The measurement model updates the belief using sensor measurements. It computes the distribution

$$P(x_t, l_1, \dots, l_N | z_t, x_{t-1}, l_1, \dots, l_N, u_t) \quad (4.5)$$

Typically, the prediction stage entails modeling the robot's motion with a motion model, such as an odometry or velocity-based model. Based on the prior stance

at time step $t - 1$ and the control input u_t , the motion model predicts the robot's pose at time step t . Sensor measurements are incorporated into the belief during the update stage. It employs a measuring model associating the observed landmarks with the robot's attitude. Given the current pose and landmark positions, the measurement model computes the likelihood of witnessing the sensor measurements.

SLAM aims to find the best a posteriori (MAP) estimate of the robot's pose and landmark placements. This estimate is derived by iteratively updating the belief in response to fresh measurements. Equations in SLAM can vary depending on the specific formulation and algorithm used. However, here are some commonly used equations:

1. Prediction Step (Motion Model)

$$x_t = f(x_{t-1}, u_t) + w_t \quad (4.6)$$

Here, $f()$ represents the motion model that predicts the robot's pose, and w_t is the process noise.

2. Update Step (Measurement Model)

$$z_t^k = h(x_t, l_k) + v_t^k \quad (4.7)$$

The measurement model $h()$ connects the robot's stance and landmark position to the sensor measurements. The measurement noise is represented by v_t^k .

3. Bayes' Filter Update

$$P(x_t, l_1, \dots, l_N | z_t, x_{t-1}, l_1, \dots, l_N, u_t) \propto P(z_t | x_t, l_1, \dots, l_N, u_t) P(x_t, l_1, \dots, l_N | x_{t-1}, l_1, \dots, l_N, u_t) \quad (4.8)$$

The belief update combines the prediction and measurement steps using Bayes' rule. The likelihood $P(z_t | x_t, l_1, \dots, l_N, u_t)$ represents the compatibility between the predicted measurements and the actual measurements.

To improve the accuracy of the estimated map and robot pose, SLAM algorithms frequently employ additional techniques such as data association (matching observed measurements with landmarks) and loop closure detection (detecting revisited sites).

4.1.1 Types of SLAM

There are different types of SLAM algorithms, each with its own approach and equations. Let's explore the most common types:

1. **EKF-SLAM (Extended Kalman Filter SLAM):** EKF-SLAM is a popular SLAM technique that combines state estimation with feature-based mapping using the Extended Kalman Filter (EKF) [31]. It is assumed that a robot having range and bearing sensors is used to detect landmarks in the area. Landmarks, such as visible features or radio beacons, can be anything with a prominent position.

EKF-SLAM's state estimation consists of two components:

- (a) **Estimation of robot pose:** This provides the position (x, y) and orientation (θ) of the robot in the world frame.
- (b) **Landmark locations:** The world frame positions of detected landmarks.

The EKF-SLAM algorithm is divided into two steps:

- (a) **Prediction:** The algorithm anticipates the robot's next pose using motion models and control inputs.
- (b) **Update:** The algorithm updates the predicted coordinates of the landmarks and the robot's pose using sensor measurements (range and bearing).

Advantages

- EKF-SLAM is a well-known method with a solid theoretical base.
- It is capable of handling nonlinear motion and sensor models.
- The method estimates the robot's stance and landmark positions consistently.
- In comparison to other SLAM algorithms, EKF-SLAM is computationally efficient.
- It is appropriate for settings with a reasonable amount of landmarks.

Disadvantages

- When operating in highly nonlinear situations, EKF-SLAM implies linearization of motion and sensor models, which may result in inaccuracies.

- The algorithm presumes that all landmarks are viewed at each time step, which may not be possible in practice.
- EKF-SLAM is vulnerable to the data association problem, which involves appropriately associating measurements with landmarks, especially in cases with comparable or ambiguous features.
- Because of the buildup of linearization mistakes, the algorithm's performance can decrease in large-scale systems.

Typical use cases: EKF-SLAM is frequently utilized in applications requiring real-time performance and moderate-sized environments. It is appropriate for situations with a reasonable number of landmarks, range, and bearing sensors. Some typical use cases are as follows:

- **Mobile Robotics:** EKF-SLAM can navigate and map indoor and outdoor areas in mobile robots [32] such as autonomous cars or drones. It allows the robot to determine its position and map its surroundings using onboard sensors such as LiDAR or cameras.
 - **Augmented Reality:** EKF-SLAM can be used in augmented reality applications to correctly overlay virtual items on the actual world. It provides realistic augmentation and interaction with the user's surroundings by estimating the device's pose and mapping the environment as mentioned in [33].
 - **Robot Manipulation:** EKF-SLAM can help with robot manipulation tasks in which the robot must recognize and interact with things in its surroundings. It enables accurate manipulation and object recognition by creating a map of the items and calculating the robot's pose.
2. **FastSLAM:** FastSLAM is a particle filter-based SLAM technique that addresses the computational cost of EKF-SLAM by estimating the robot pose and separate EKFs for each landmark using a particle filter [34]. FastSLAM displays the map as a collection of particles containing a robot posture hypothesis and a collection of landmark estimates.

FastSLAM employs a recursive Bayesian filter framework and proceeds as follows:

- (a) **Particle prediction:** Each particle's pose is updated using motion models and control inputs.

- (b) **Measurement update:** The algorithm adjusts the particle weight based on the likelihood of the sensor measurements for each particle.
- (c) **Resampling:** Higher-weight particles are duplicated, while lower-weight particles are discarded. This stage assists in directing computing resources toward more likely hypotheses.

Advantages

- FastSLAM solves the data association problem by modeling the map as a set of particles, which is more robust in the circumstances with ambiguous features or many hypotheses.
- It can work in environments with a high density of landmarks.
- FastSLAM is computationally efficient since the particle filter method enables computation parallelization.
- The approach generates a complete posterior distribution of the robot's pose and landmark positions, allowing uncertainty to be estimated.

Disadvantages

- FastSLAM employs a particle filter, which requires a large number of particles to represent the posterior distribution accurately. As a result, the technique may necessitate more computational resources than EKF-SLAM.
- FastSLAM performance can suffer when the environment varies significantly or when the number of landmarks is too high to be efficiently represented by particles.
- Because the typical FastSLAM formulation does not explicitly reflect loop closures, it may struggle with scenarios with significant loop closures.

Typical use cases: FastSLAM works well in situations with a high density of landmarks or scenarios with ambiguous features. It excels when the data association problem is complex and requires handling several hypotheses. Some typical use cases are as follows:

- **Robot Exploration:** FastSLAM can be utilized in exploration missions where a robot must traverse and map an unknown environment autonomously. The particle filtering method enables the robot to manage uncertain and ambiguous measurements, allowing for fast exploration and map construction.

- **Autonomous Mapping:** FastSLAM is appropriate for autonomous mapping applications that aim to map an environment using onboard sensors. It can be utilized when a robot needs to map large-scale settings with various landmarks, such as environmental monitoring or search and rescue missions.
 - **Humanoid Robotics:** FastSLAM can be used in humanoid robotics scenarios where the robot interacts with a changing and dynamic environment [35]. It allows the robot to precisely estimate its stance while tracking and mapping dynamic objects or landmarks.
3. **GraphSLAM:** GraphSLAM considers SLAM to be a graph optimization problem. The environment is represented as a graph, with nodes representing robot postures and landmarks and edges representing constraints between them (for example, mobility and measurement constraints) [36].

GraphSLAM seeks the most likely approximation of the graph given the observed observations. It solves the optimization problem by minimizing an objective function. It is frequently written as the sum of error terms indicating the disparities between actual and anticipated measurements based on estimated poses and landmarks.

Nonlinear optimization approaches, such as the Gauss-Newton or Levenberg-Marquardt algorithms, are commonly used to minimize the objective function.

Advantages

- GraphSLAM provides a versatile framework for representing and solving SLAM issues via graph optimization.
- It can explicitly handle loop closures, allowing for more accurate and robust mapping in contexts with frequent revisits.
- Because the optimization process analyses all constraints simultaneously, the technique produces a global map representation.
- GraphSLAM can handle a variety of sensor modalities as well as non-Gaussian noise models.

Disadvantages

- The formulation and solution of a nonlinear optimization problem are required for GraphSLAM, which can be computationally expensive, especially for large-scale maps.

- It presumes that the motion and measurement models are known and accurate enough, which may only sometimes be true in real-world circumstances.
- In dynamic contexts, when the assumptions of static landmarks and consistent motion models are violated, GraphSLAM may struggle.
- GraphSLAM's success depends on accurate graph setup and effective handling of loop closures.

Typical Use Cases: GraphSLAM is commonly utilized in scenarios requiring loop closures and global map consistency [37]. It excels in contexts with frequent revisits and requires a thorough map representation. Some typical use cases are as follows:

- **Autonomous Driving:** GraphSLAM is often utilized in self-driving applications when precise localization and mapping are required. It allows the vehicle to have a globally consistent map while dealing with loop closures caused by repeated journeys or returning sites.
- **Robotics Simultaneous Localization and Mapping:** In robotics SLAM research and development, GraphSLAM is widely employed. It enables researchers to investigate and create novel methods and approaches for simultaneous localization and mapping while ensuring global consistency in map representation [38].
- **Large-scale Mapping:** GraphSLAM is well-suited for mapping large-scale settings with many landmarks and frequent loop closures. It is used in outdoor mapping, construction site monitoring, and urban planning.

4.1.2 Challenges of SLAM

SLAM encounters numerous issues due to the inherent uncertainties in robot perception and mobility. Sensor noise and calibration, data association, computational complexity, and dynamic settings are the key areas where these issues can be found. Let us go over each challenge in depth.

1. **Sensor Noise and Calibration:** As explained in dealing with sensor noise and accurately calibrating sensors is one of the critical issues in SLAM [39]. Robotic sensors, such as cameras and range finders, introduce noise into readings, leading to errors in calculating the robot's position and building

the map. Gaussian distributions can be used to model the noise. x_t represents the robot's condition at time t , and z_t represents the measurements at time t . The following equation shows the link between the true state and noisy measurements:

$$z_t = h(x_t) + v_t \quad (4.9)$$

Where $h()$ is the sensor model, and v_t represents the measurement noise.

2. **Data Association:** The process of appropriately linking measurements with map features is called data association [40]. This difficulty increases when the environment contains recurrent or confusing elements. The data association problem can be expressed as a correspondence problem to determine the proper relationships between measurements and map characteristics. Probabilities can be used to represent the data association problem. Let z_t and x_t be the measurements and state, respectively, at time t . The likelihood of linking a measurement z_t with a map feature m_i is given by:

$$P(\text{Association}|z_t, x_t, m_i) \quad (4.10)$$

3. **Computational Complexity [41]:** Due to the vast amount of data to analyze and the complexity of the underlying optimization problems, SLAM techniques frequently necessitate substantial computational resources. Given all measurements in these optimization situations, the combined probability of the robot's trajectory and the map is often maximized. The computational complexity grows as the number of features, map size, and trajectory length grows. This can become computationally difficult in large-scale ecosystems. The computational complexity can be defined as $O(NK^2)$ in terms of the number of map features (N) and poses (K).
4. **Dynamic Environments:** SLAM algorithms are typically built for static situations, where it is assumed that the scene does not change over time. Moving objects, however, might pose problems for SLAM in dynamic situations. Moving objects contradict the static world assumption, resulting in mapping and localization issues. As suggested in the dynamics of the environment can be modeled by incorporating motion models for moving objects into the SLAM framework [42].

4.2 Methodology

We developed a system that can perform real-time localization and mapping while adhering to size, weight, and power consumption constraints. Furthermore, another challenge is the limited range and resolution of lightweight LiDAR sensors. Due to their reduced size and weight, these sensors often have a lower detection range and less precise measurements compared to their heavier counterparts. This can lead to difficulties in accurately perceiving the environment, especially in scenarios that require long-range perception or fine-grained mapping.

This section describes the suggested method in depth. As shown in Fig. 4.1, the system is separated into three key modules: feature extraction, odometry estimation, and probability map production. First, the rotation invariant feature extraction method is demonstrated, followed by odometry estimation using local feature matching. Finally, we demonstrate how to generate a probability map and recreate a scenario.

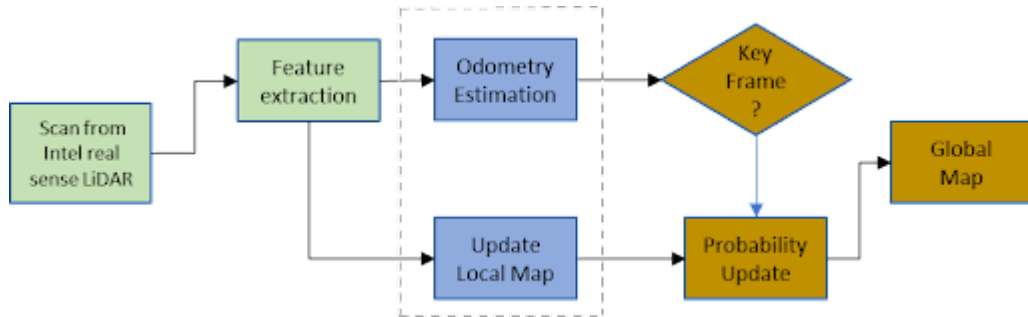


Figure 4.1: Method Overview

4.2.1 Feature Extraction

Solid-state LiDAR technology has various advantages over mechanical LiDAR, including better resolution and frequency of update. However, improved data quality comes at the expense of greater computational requirements for processing. In this research, we offer an approach that uses edge and planar matching techniques, which are more computationally efficient than previous methods such as LOAM (LiDAR Odometry and Mapping).

To prepare the data for processing, we first use the computed distance to detect and remove noisy areas. Noisy zones are frequently found within the LiDAR's maximum detection range when the reflected energy is low, resulting in less accurate results. We improve the overall accuracy of the upcoming analysis by filtering out these noisy sites beforehand.

A LiDAR scan often yields an unsorted Point cloud that can be difficult to work with directly. The Point cloud is projected into a 2D point matrix to address this. This transformation makes it easier to compute edge and planar features. Each point in the LiDAR scan, denoted by $p_i = (x_i, y_i, z_i)$, has a vertical and horizontal angle, denoted by α_i and θ_i , respectively.

$$\alpha_i = \arctan\left(\frac{y_i}{x_i}\right) \quad (4.11)$$

$$\theta_i = \arctan\left(\frac{z_i}{x_i}\right) \quad (4.12)$$

The Point cloud is then segmented by equally dividing the vertical detection range $[\alpha_{min}, \alpha_{max}]$ and horizontal detection range $[\theta_{min}, \theta_{max}]$ into M and N sectors, where $\alpha_{min}, \alpha_{max}, \theta_{min}, \theta_{max}$ are the sensor specifications' minimum vertical angle, maximum vertical angle, minimum horizontal angle, and maximum horizontal angle. As a result, the Point cloud is divided into MxN cells. In [9] M and N are chosen as half of the total points in a single direction for a solid-state LiDAR with vertical angular α_r resolution and horizontal θ_r resolution, i.e., $M = \frac{\alpha_{max} - \alpha_{min}}{2 \times \alpha_r}$ and $N = \frac{\theta_{max} - \theta_{min}}{2 \times \theta_r}$. In each cell (m, n) , where $n \in [1, N], m \in [1, M]$ and the symbol $[1, M]$ represents $\{1, 2, \dots, M\}$, we calculate the mean measurement $\mathbf{p}_k^{(m,n)}$ by finding the geometric center of all points in each cell.

4.2.2 Odometry Estimation

The process of estimating the robot location is termed as Odometry estimation. current pose $T_k \in SE(3)$ in global coordinates based on the previous laser scan P_1, P_2, \dots, P_{k-1} . Historically, the trajectory is computed using either a scan-to-map or a scan-to-scan match. The fastest frame from the previous scan is aligned to the current frame in scan-to-scan match. A single laser scan contains less surrounding information as compared to a local map. In the long run, this creates drift. As a result, we apply the scan to map match to boost performance. To mitigate the computational costs, the sliding window method is used. The planar and edge features from surrounding frames are used to construct the local feature maps. We define the local map M_k for the current input P_k as $M_k = (P_{k-1}, P_{k-2}, \dots, P_{k-q})$, where q is the number of frames utilized to generate the local map. Algorithm 1 describes the method of iterative odometry estimation.

Algorithm 1: Pose estimation for Solid-state LiDAR

Data: Current Scan P_k , Robot Trajectory $\mathbf{T}_{1:k-1}$
Result: Current pose \mathbf{T}_k
if Not Initialized **then**
 | $T_0 \leftarrow 0$;
end
Calculate local smoothness and perform feature extraction;
Compute initial alignment $\mathbf{T}_k^0 \leftarrow \mathbf{T}_{k-1} \mathbf{T}_{k-2}^{-1} \mathbf{T}_{k-1}$;
while Pose estimation is not optimized **do**
 instructions;
 for each $p_k \in P_k$ **do**
 Transform to map coordinate $\mathbf{p}_k \leftarrow \mathbf{T}_k^{i-1} \mathbf{p}_k$
 if \mathbf{p}_k is an edge **then**
 | add edge cost factor;
 end
 if \mathbf{p}_k is a plane **then**
 | add edge cost factor;
 end
 if nonlinear optimization converges **then**
 | Compute \mathbf{T}_k^i ;
 end
 end
 Update current scan and remove the oldest scan from local map;
 Return current pose \mathbf{T}_k ;
end

4.2.3 Probablity Map Construction

The global map is frequently enormous, and updating it with each frame could be more computationally efficient. As a result, we solely update and recreate the map using critical frames. The key frames are chosen using the following criteria:

1. If the robot's displacement is significant enough (i.e., larger than a pre-defined threshold).
2. If the change in rotation angle (including roll, pitch, and yaw angle change) is significant.
3. If the time elapsed exceeds a specific period.

In practice, the sensor's field of view determines the rotation and translation thresholds, while the processor's processing power determines the minimum update rate. The global map is built using an octree to improve search efficiency.

Searching for a given node in an octree of depth n only takes $O(\log n)$ computational complexity, which can considerably reduce mapping cost [8]. We use $P(n|z_{1:t})$ to present the probability of the existence of an object for each cell in an octree [9]:

$$P(n | z_{1:t}) = \left[1 + \frac{1 - P(n | z_t)}{P(n | z_t)} \cdot \frac{1 - P(n | z_{1:t-1})}{P(n | z_{1:t-1})} \cdot \frac{P(n)}{1 - P(n)} \right]^{-1}, \quad (4.13)$$

Here, $P(n)$ is the prior probability, which is set to 0.5 in the case of an unknown region, z_t is the current measurement and $z_{1:t-1}$ is the historical measurement from key frames.

4.2.4 Novelty of the Proposed Methodology

We have taken inspiration from [9] and developed a light weight SLAM algorithm. Our method mitigates the use of the IMU sensor. Moreover, we have made changes in the Feature Extraction used in [9] In order to improve results we have taken $M = \frac{\alpha_{\max} - \alpha_{\min}}{\alpha_r}$ and $N = \frac{\theta_{\max} - \theta_{\min}}{\theta_r}$. Due to increase in value of M and N the density of points in resultant 3D mapping has increased whereas the time taken by algorithm has marginally gone up. Moreover, We have not used IMU sensor which was used in [9]. Also, [9] uses warehouse robot to conduct experiments whereas we have used handheld system to conduct the experiments in order to make it more realistic.

CHAPTER 5

Experimental Verification

5.1 Experimental Setup

This chapter focuses on the experiments and the experimental setup used to conduct various experiments. We begin by describing the procedure of the experiments, followed by explaining the physical experimental setup, and finally by explaining the internal working of the proposed system.

We have conducted several experiments with our dataset on both versions of the code: the original and our code. We have recorded the entire Robotics Lab using an Intel Realsense L515 LiDAR camera in different light situations and different translational and rotational speeds. We have used the ROS Noetic version of ROS with Ubuntu 20.04 LTS. Rviz was used to display the final output of the experiments. Figure 5.1 depicts the experimental setup which was used for conducting the experiments. As we can see a window depicting ROS Master which connects all the ROS nodes and maintains the list of all the topics and nodes. We can also see raw input Point cloud as well as camera input in Rviz.

The internal working and how data flows through various components are depicted in Figure 5.2. The Intel Realsense L515 LiDAR camera captures the RGB camera feed along with the 3D point clouds. The raw data is then processed in the `ssl_slam` node, which generates the odometry and 3D mapping using raw data. Finally, the Rviz node displays the processed final output in a user-friendly manner. In ROS, data processing involves a pipeline of nodes performing specific data tasks. Let us explore how data flows and is processed in the proposed framework, which includes a LiDAR node, a SLAM node, and an output displaying(Rviz) node.

1. **LiDAR Node:** The LiDAR node captures images from an Intel Realsense L515 LiDAR camera. It publishes these images as messages on a specific topic, such as `"/camera/image."` The camera node acts as a publishing node, continuously sending image data to the topic.

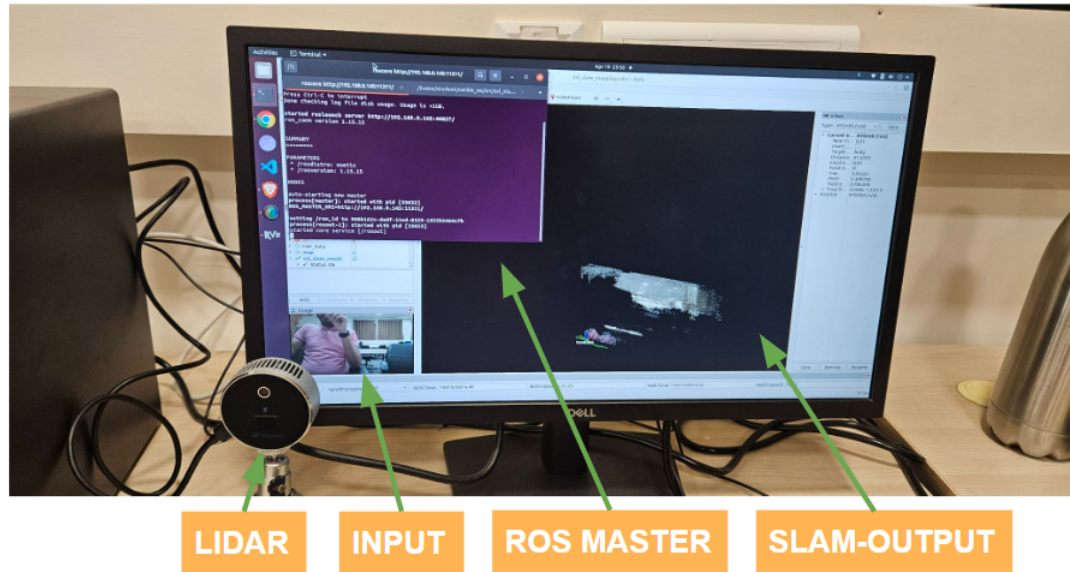


Figure 5.1: Physical experimental setup.

2. **SLAM Node:** The SLAM processing node subscribes to the `"/camera/image"` topic to receive the image data along with 3D point clouds published by the camera node. This node performs various image processing operations on the received images.
 - (a) **Subscribing to the Topic:** The SLAM node creates a subscriber object, specifying the topic (`"/camera/image"`) and the image message type. It registers a callback function invoked whenever a new image message is published on the topic.
 - (b) **SLAM Operations:** Inside the callback function, the SLAM node retrieves the received image data and 3D point clouds and applies various algorithms or operations to process the image. This can include image filtering, feature extraction, object detection, or image manipulation or analysis.
 - (c) **Generating Processed Image:** After the SLAM operations are performed, the SLAM node generates a new image message containing the processed data. This new message can have a different topic, such as `"/ssl_slam_mapping,"` `"ssl_slam_mapping_odometry,"` and a corresponding message type.
 - (d) **Publishing Processed Image:** The image processing node creates a publisher object for the `"/ssl_slam_mapping,"` `"ssl_slam_mapping_odometry,"` topic and publishes the processed image message. Other nodes interested in the processed image can subscribe to this topic.

3. **Output Displaying(Rviz) Node:** The output displaying node subscribes to the `"/ssl_slam_mapping"` and `"ssl_slam_mapping_odometry"` topics to receive the processed image messages generated by the image processing node. It is responsible for displaying the processed images in a user-friendly format, such as on a graphical user interface (GUI) or a screen.
 - (a) **Subscribing to the Topic:** The output displaying node creates a subscriber object, specifying the `"/ssl_slam_mapping"` and `"ssl_slam_mapping_odometry"` topics and the corresponding message type. It registers a callback function triggered whenever a new processed image message is published on the topic.
 - (b) **Displaying the Processed Image:** Inside the callback function, the output displaying node retrieves the processed image data from the received message and displays it on the output device, such as a screen or GUI. The specific implementation may vary depending on the display mechanism used.

By connecting these nodes, the data flow from the LiDAR node to the SLAM node and finally to the output displaying(Rviz) node. Each node performs its designated task, enabling a pipeline for image capture, processing, and display in a ROS system. This modular approach allows flexibility and scalability in designing complex robotic systems with multiple interconnected nodes.

We have conducted two types of experiments on handheld devices:

1. **Real-time:** For this, we used direct input from LiDAR as input for SLAM. It gives real-time mapping along with the estimated path of the bot.
2. **Recorded ROS bag file:** For this case, we recorded the data in the ROS bag file and then used the ROS bag file as input for the SLAM algorithm.

The results in both cases are identical; hence we have conducted all the experiments on the recorded ROS bag files, described in detail in the next chapter.

5.2 3D Point Cloud Dataset Prepared Using LiDAR

This chapter describes the datasets that have been used to conduct various experiments. Initially an image is shown describing the contents of the ROS bag file and its interpretation, followed by describing the various scenarios which have been used to conduct the experiments.

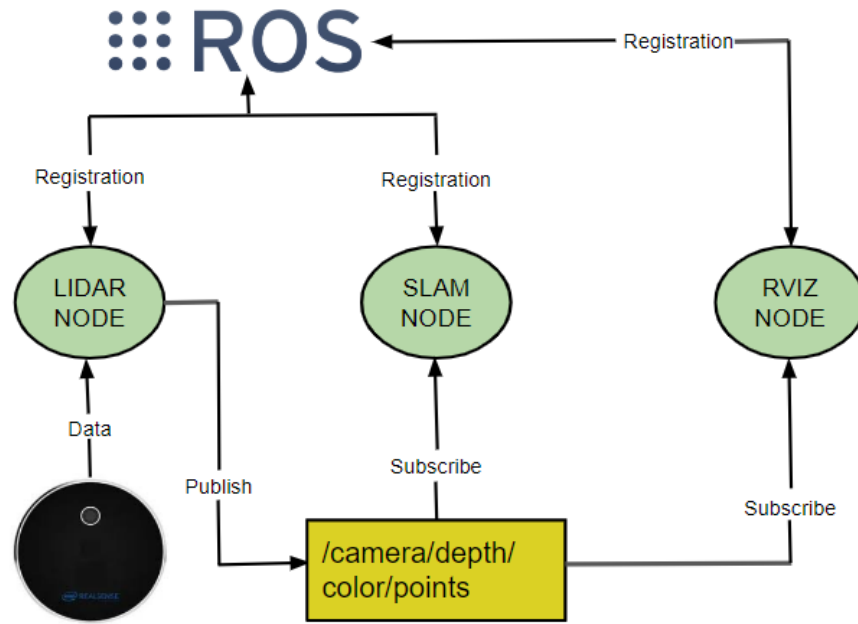


Figure 5.2: Illustrates the schematic representation of data flow of the proposed method.

A ROS bag file is a file format in ROS that is used to capture and playback data sent between nodes. It lets you preserve messages published on topics over a specific time period and playback them afterward, making it easy to analyze or reproduce data.

We recorded a ROS bag file for conducting various experiments for each scenario. Let us go over the topics that are listed in Figure 7.1, which depicts the content of one of the recorded ROS bag files and how they are documented in a ROS bag file:

1. **/camera/color/camera_info**: This item often provides information on camera calibration, such as intrinsic characteristics, distortion coefficients, and image dimensions. It contains critical metadata and is often published as a `sensor_msgs/CameraInfo` message type. When you record a ROS bag file, the messages published on this subject and their timestamps are saved in the bag file. The bag file will contain all the camera calibration data published on this topic during the recording period.
2. **/camera/color/image_raw**: This item includes the camera's raw color photos. Image data and metadata such as image dimensions, encoding format, and timestamp are often published as `sensor_msgs/Image` message type. When a ROS bag file is recorded, each message published on the /cam-

```

path:      robotics_lab_final.bag
version:   2.0
duration:  1:55s (115s)
start:     Apr 11 2023 02:11:39.18 (1681159299.18)
end:       Apr 11 2023 02:13:35.14 (1681159415.14)
size:      17.4 GB
messages:  13517
compression: none [6679/6679 chunks]
types:     sensor_msgs/CameraInfo [c9a58c1b0b154e0e6da7578cb991d214]
           sensor_msgs/Image      [060021388200f6f0f447d0fcd9c64743]
           sensor_msgs/PointCloud2 [1158d486dd51d683ce2f1be655c3c181]
topics:    /camera/color/camera_info    3351 msgs    : sensor_msgs/CameraInfo
           /camera/color/image_raw  3351 msgs    : sensor_msgs/Image
           /camera/depth/camera_info 3481 msgs    : sensor_msgs/CameraInfo
           /camera/depth/color/points 3334 msgs    : sensor_msgs/PointCloud2

```

Figure 5.3: Dataset description in ROS.

era/color/image_raw topic is saved in the bag file, together with the image data and metadata. Over time, the bag file will collect a succession of color photos.

3. **/camera/depth/camera_info:** The /camera/depth/camera_info subject, like the /camera/color/camera_info topic, provides camera calibration information for the depth sensor. It typically contains intrinsic characteristics, distortion coefficients, and depth-specific image dimensions. Typically, the messages are published as sensor_msgs/CameraInfo message types. All the camera calibration data presented on this topic will be saved during bag file recording, allowing you to obtain the depth of camera calibration information upon playback.
4. **/camera/depth/color/points:** This subject provides the depth sensor's three-dimensional Point cloud data. It gives a set of 3D points as well as the RGB color information for each of them. The sensor_msgs/PointCloud2 message type is commonly used to publish Point cloud data. When a ROS bag file is recorded, the messages published on the /camera/depth/color/points topic are preserved with the Point cloud data and its accompanying metadata. The bag file will save the Point cloud data, allowing you to replay and process it later.

Overall, a ROS bag file comprising the mentioned topics will save the messages published on each topic over the recording period. Camera calibration data, raw color photos, depth camera calibration information, and 3D Point cloud data are all included. When replayed, the bag file allows you to extract and analyze the

recorded data and perform things like debugging, testing, and designing algorithms without actual equipment. We have three different types of datasets, which are as follows:

1. **Standard Lighting:** For this case, we turned on all the lights in the Robotics Lab and recorded the data using the setup described in the previous section.
2. **Less Lighting:** For this case, we turned off all the lights except three in the Robotics Lab and recorded the data using the setup described in the previous section.
3. **No Light:** For this case, we turned off all the lights in the Robotics Lab and recorded the data using the setup described in the previous section.

We have a precise and accurate path on which the LiDAR camera had moved for the first two cases, whereas for the last case, as it was very dark, the actual path's ground truth is less accurate than in previous cases.

5.3 Result and Discussion

This chapter compares the results of various experiments on our dataset as well as the dataset provided by [9] using our proposed method and the method used by [9]. We have performed several experiments to compare and contrast our results with the existing method in order to evaluate our method. In this chapter, we will compare the results obtained on our dataset as well a demo dataset using two versions of code: the original and our code. Figure 5.4 and 5.5 depicts the result on a demo dataset of a warehouse that was used in [9]. The ROS bag file contains similar data types, as shown in Figure 5.3. We observed that the Figure 5.5 has more detailed features as well as a more dense 3D mapping of the warehouse compared to Figure 5.4.

Figure 5.6 shows the results of the original algorithm used in [9]. First part shows the top view in which we can see the 3D shape of the Robotics Lab, whereas the later part shows the estimated odometry of the Robot. From Figure 5.6(a), we conclude that the original algorithm works properly and gives an almost perfect cuboidal shape of the Robotics Lab. Figure 5.b(b) shows that the SLAM algorithm accurately identifies the Odometry of the Robot. Similarly, Figure 5.7 proves the effectiveness of our algorithm in Localization and Mapping. The mapping of 5.6 is denser as compared to 5.5, which further proves that our method gives more precise results.

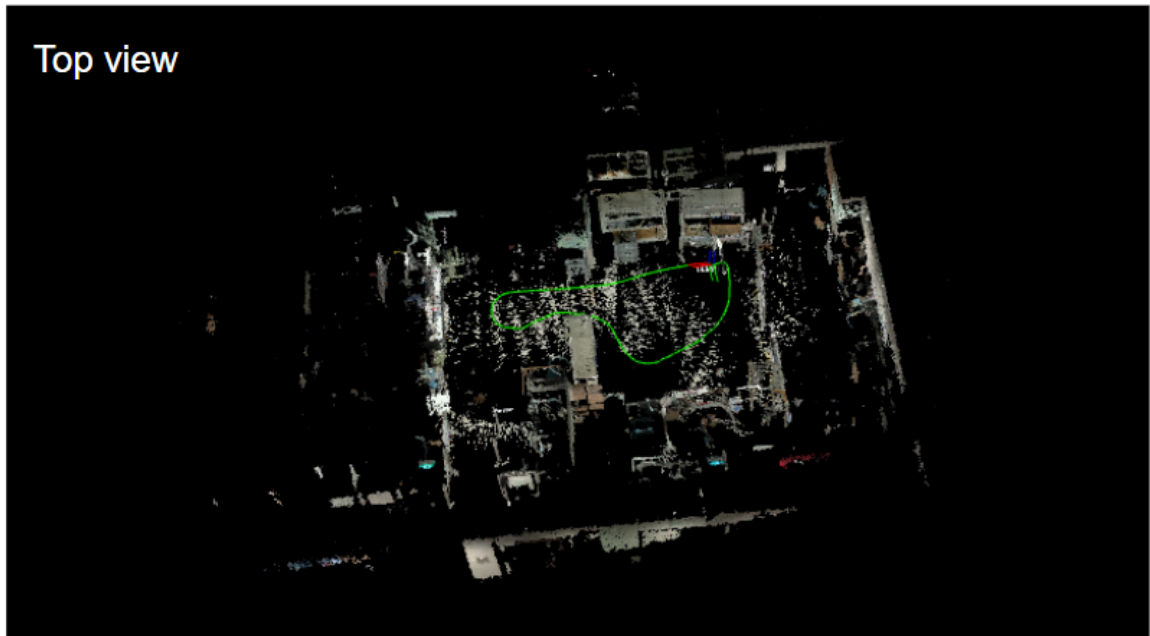


Figure 5.4: Results obtained with existing algorithm applied on demo dataset [9].

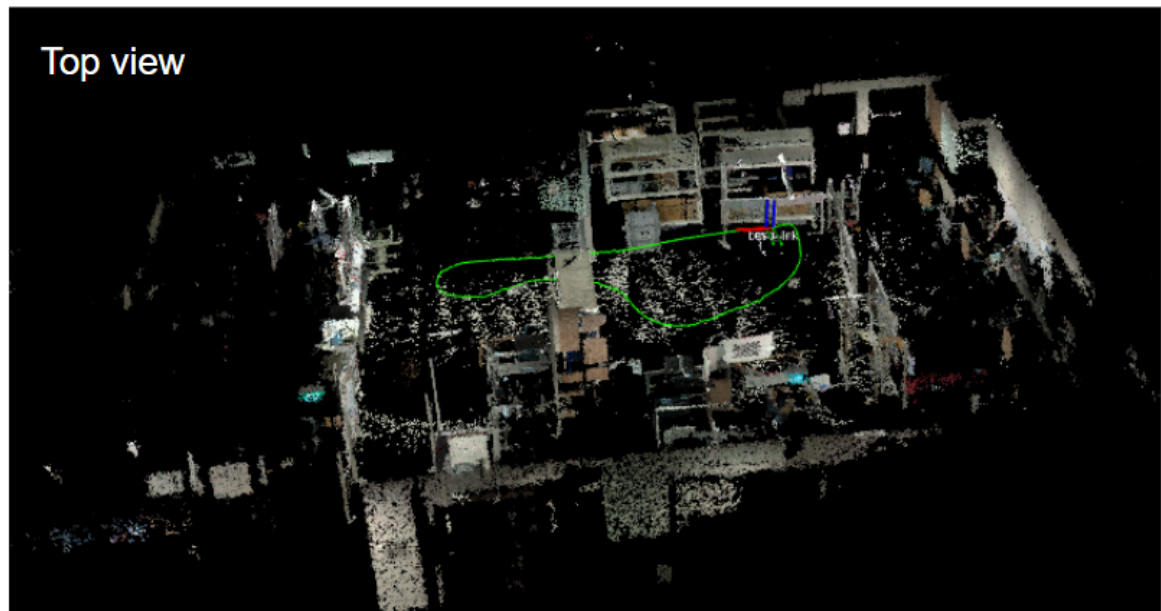
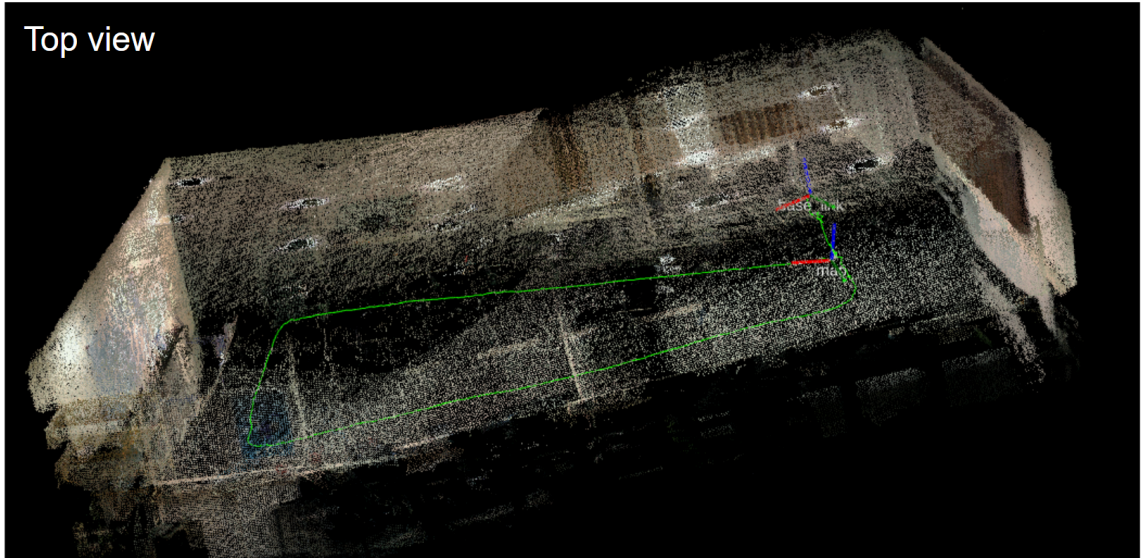
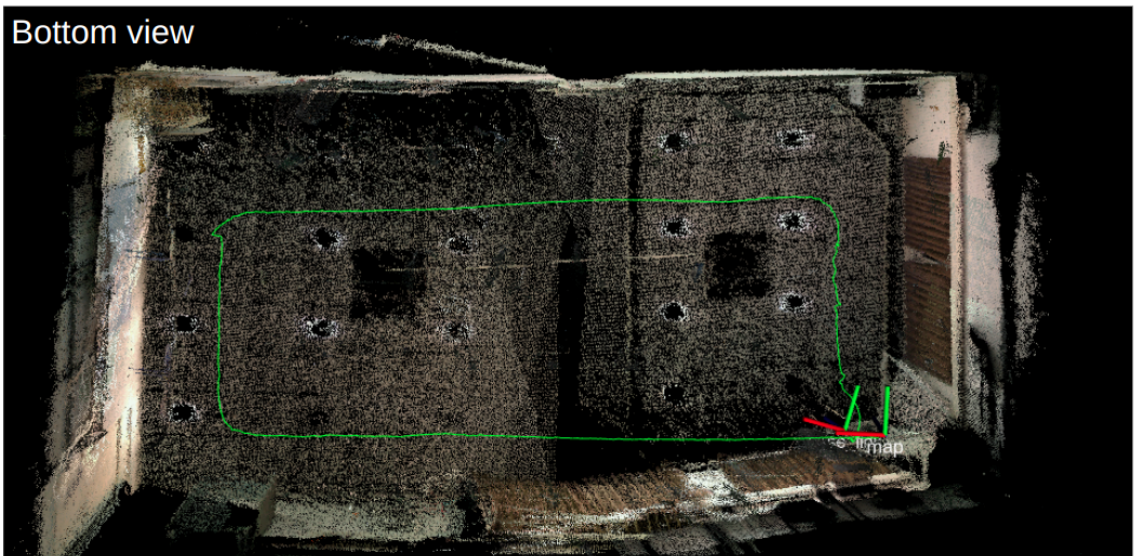


Figure 5.5: Results obtained with our improved algorithm applied on demo dataset.

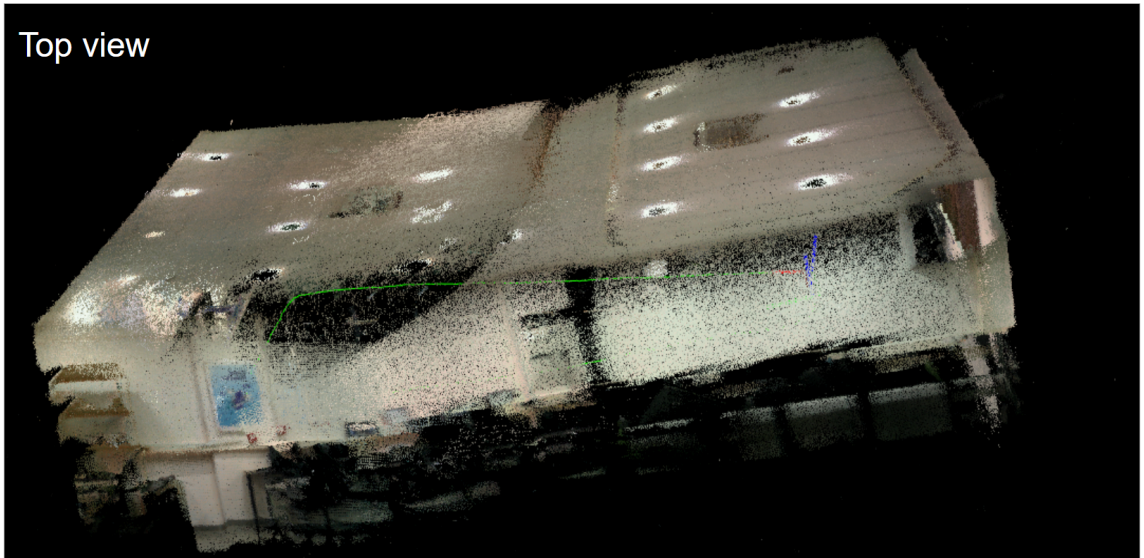


(a)

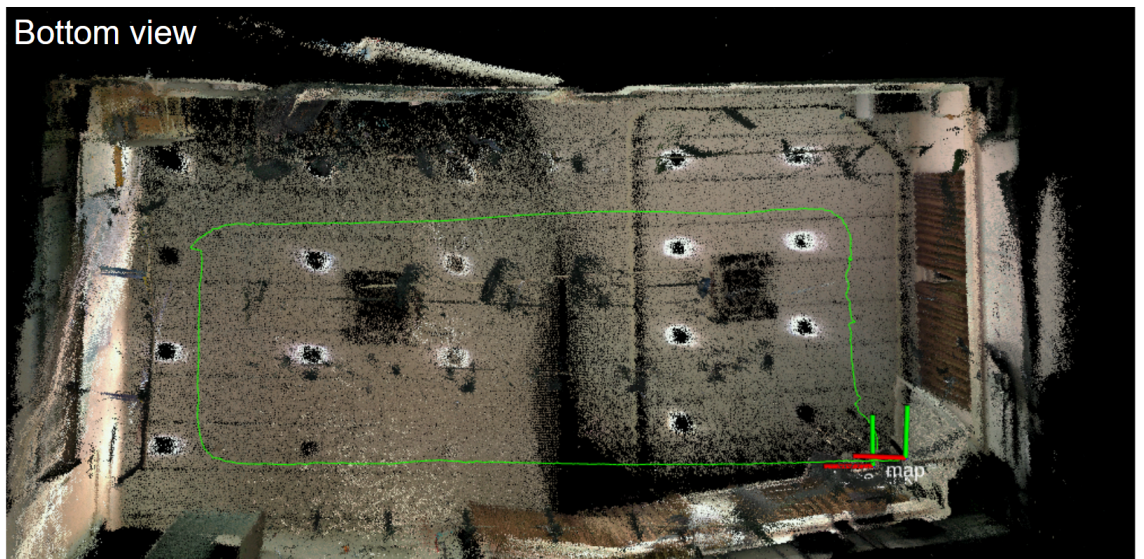


(b)

Figure 5.6: Results obtained with the algorithm [9] used on our bright dataset.



(a)



(b)

Figure 5.7: Results obtained with improve algorithm used on our bright dataset.

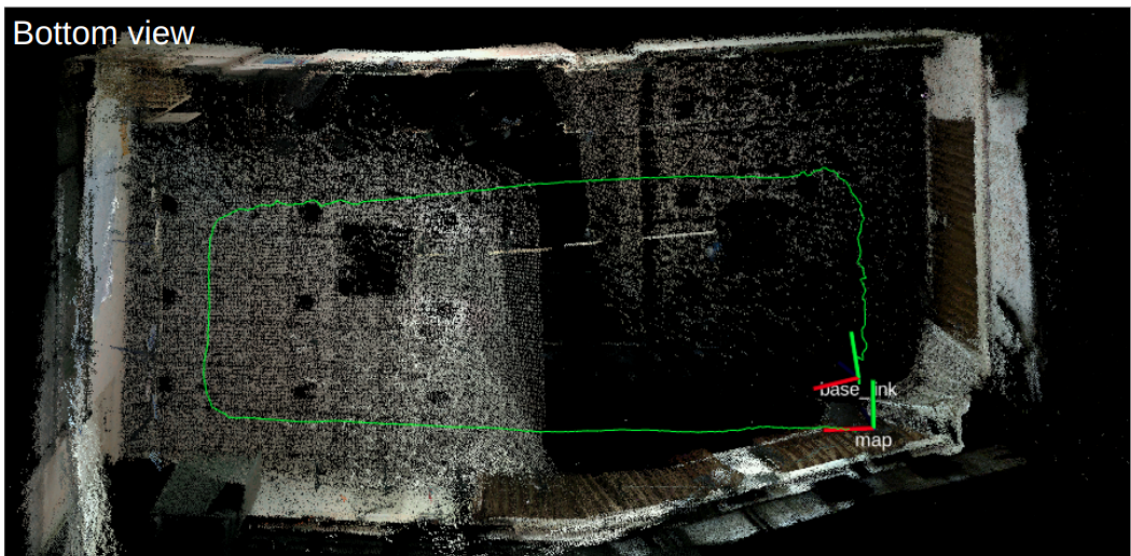
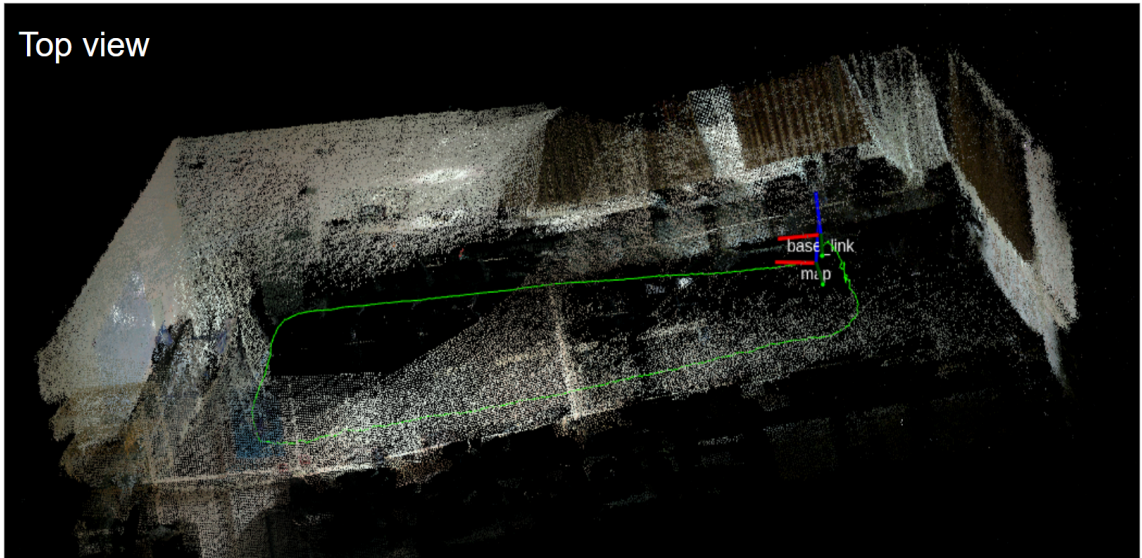


Figure 5.8: Results obtained with existing algorithm used on our dark dataset [9].

Figure 5.8 shows the result obtained on our dark data dataset, which was recorded to simulate an environment with little less lighting. Figure 5.8(b) shows that the Odometry is unaffected despite insufficient lighting, and Figure 5.8(a) shows that the mapping is less dense as compared to Figure 5.8(b). Due to the scarcity of light, points reflected in the LiDAR camera are fewer as compared to the previous case, Hence a less dense mapping. Figure 5.9 shows the result obtained on our dark data dataset using our algorithm, which was recorded to simulate an environment with little less lighting. Figure 5.9(b) shows that the Odometry is unaffected despite insufficient lighting, and Figure 5.9(a) shows that the mapping is denser as compared to Figure 5.9(b).

Figures 5.10 and 5.11 show the results of our completely dark dataset on the [9]'s algorithm and our improved algorithm, respectively. This dataset was recorded in a completely dark environment; hence the trajectory is a little irregular compared to previous cases. Due to the scarcity of light, walking on the path we were supposed to walk on was impossible. Despite all these limitations, the Odometry estimation is nearly perfect, further solidifying the SLAM algorithm's effectiveness in mapping. Although in Figures 5.10 and 5.11, we are not able to clearly see the 3D mapping, we observed a very light mapping in Rviz. This proves the effectiveness of the LiDAR camera as well as our SLAM algorithm in an entirely dark environment. A graph comparing the actual path and the estimated path by our SLAM algorithm, which the LiDAR camera followed, is shown in Figure 5.12. The actual path was estimated with the help of the tiles in the Robotic Lab @ DA-IICT. We tried to follow the same tiles every time we recorded the data. Then we measured the distance of tiles from the walls and edges to get the actual path. Then we scaled the actual path to the same scale as the estimated path. To get the estimated path, we saved the results of the `ssl_slam\Odometry` node in a ROS bag file. Followed by extracting and plotting the co-ordinates using a python script. Figure 5.12 shows how accurate our localization is. We clearly identified that the estimated path almost overlaps the actual path. The slight deviation is due to human error while recording the dataset, as we cannot walk in a straight line.

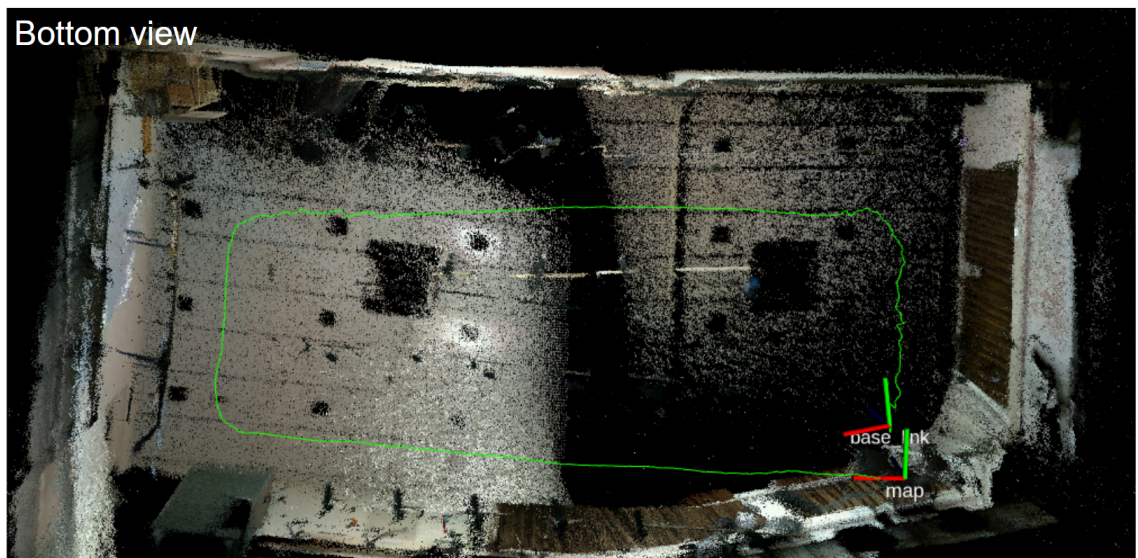
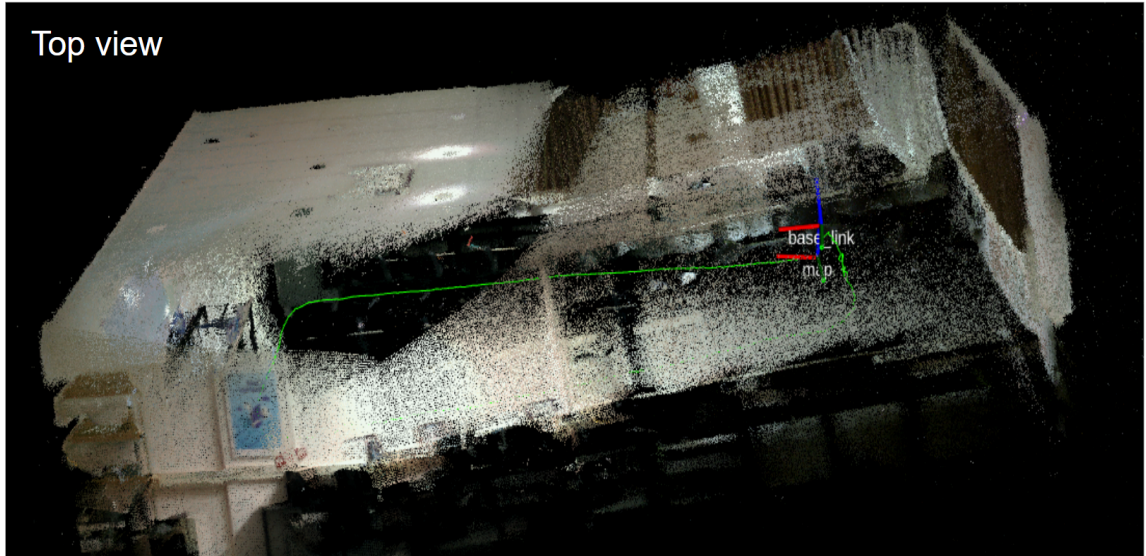


Figure 5.9: Results obtained with improved algorithm used on our dark dataset.

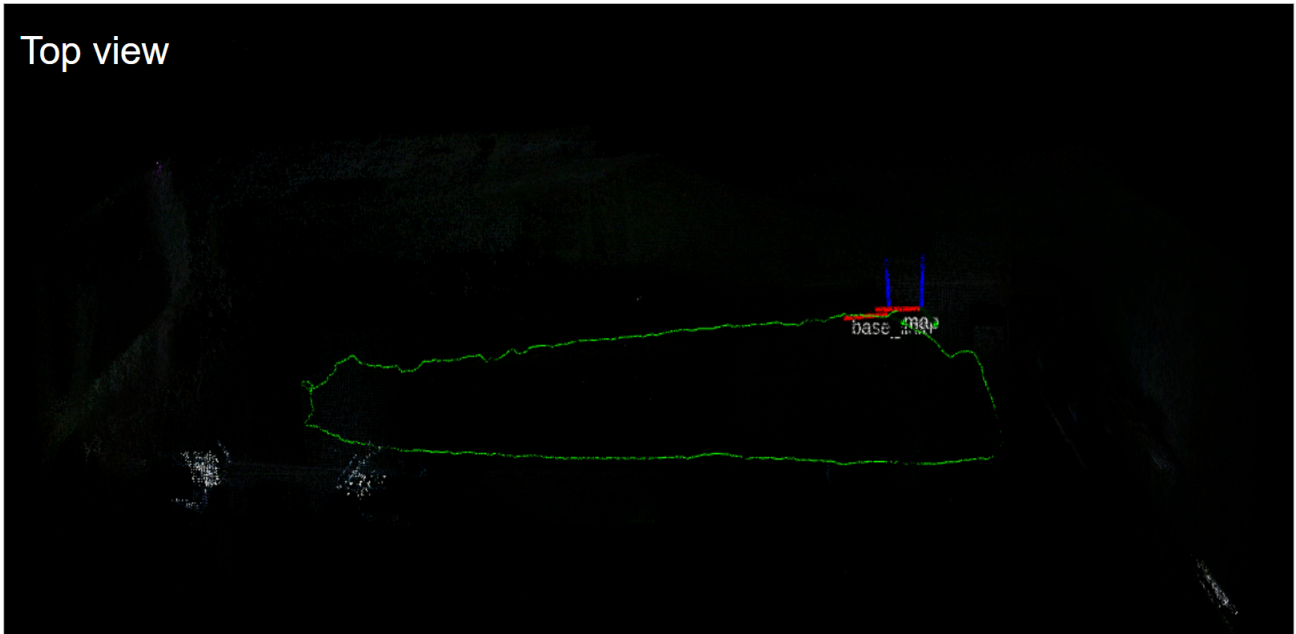


Figure 5.10: Results obtained with existing algorithm used [9] applied on our completely dark dataset.

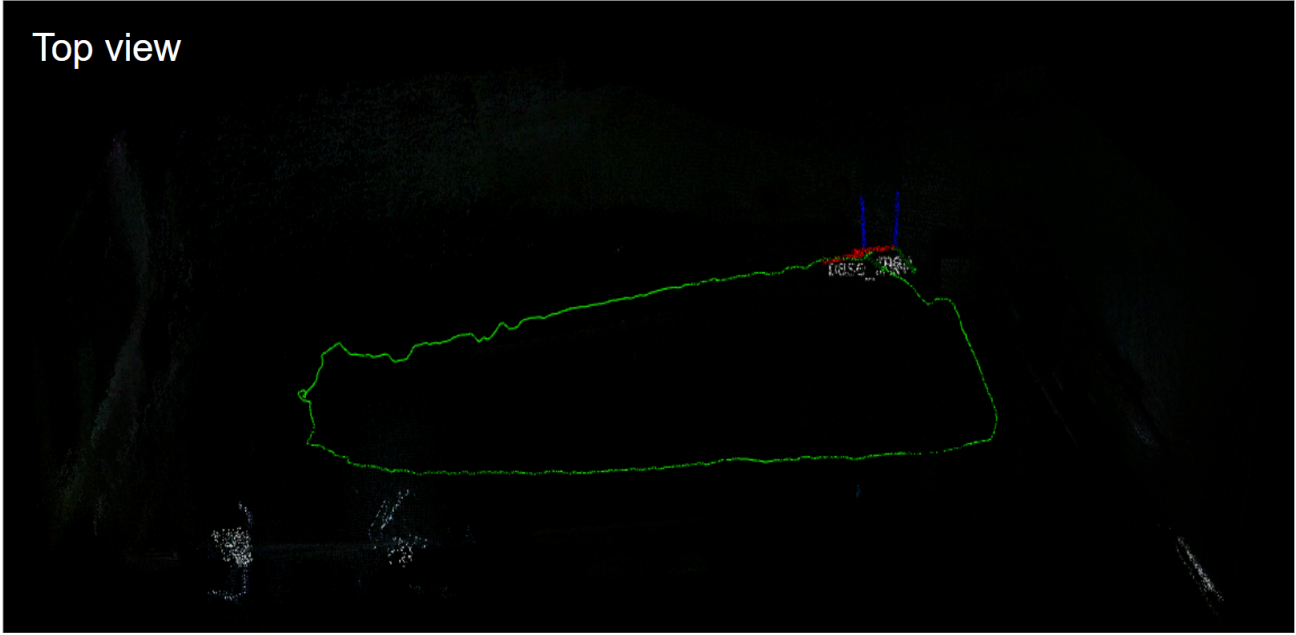


Figure 5.11: Results obtained with improved algorithm used on our completely dark dataset.

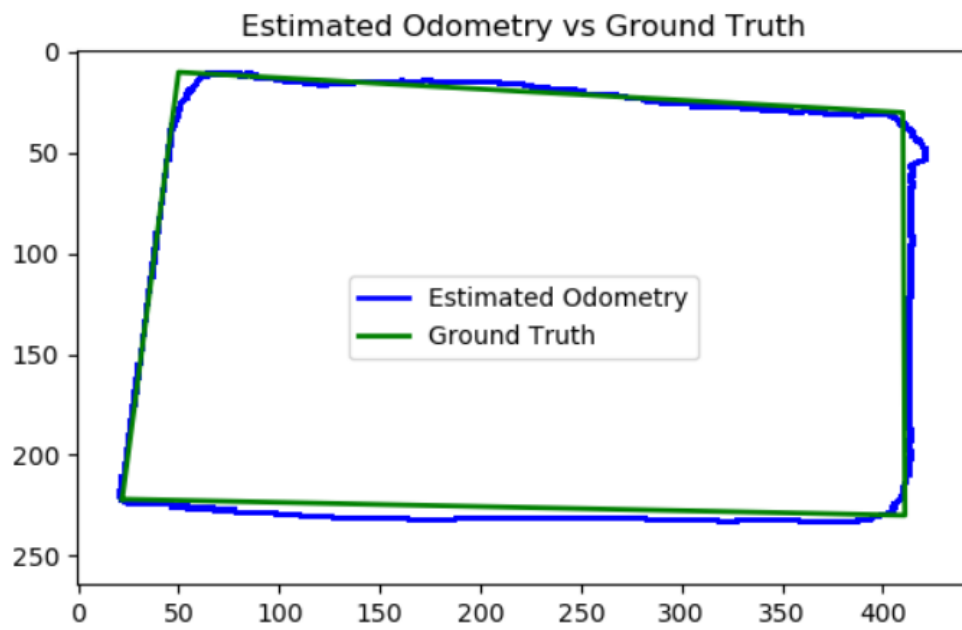


Figure 5.12: Estimated odometry vs. actual odometry.

CHAPTER 6

Conclusions

We used a handheld LiDAR sensor to construct and test a LOAM-based SLAM algorithm in a simulated environment. The purpose was to precisely estimate the robot's trajectory and create an environment map. Compared to the existing algorithm, we have not used an IMU sensor as well as a handheld device rather than a mobile bot. Moreover, we have also modified the algorithm to give better results.

The SLAM algorithm produced very encouraging results. Throughout the experiment, the projected robot trajectory displayed negligible drift and accurately mirrored the robot's actual motion. The smooth trajectory, with no abrupt jumps or irregularities, showed that the optimization procedure effectively minimized errors. Furthermore, the created map has a high level of detail and accuracy. Structures in the area, such as walls, corners, and barriers, were precisely captured and depicted on the map. The map displayed a consistent and cohesive environment representation that corresponded well to ground truth measurements.

The algorithm demonstrated robustness in dealing with a variety of challenging conditions, such as loop closures and sensor noise. It effectively detected loop closures, resulting in loop constraint optimizations that enhanced the map's accuracy and consistency. Noise in sensor measurements was reduced, yielding clean and precise mapping findings. Furthermore, the SLAM algorithm's computing speed was remarkable. The method runs in real-time, allowing for live map construction and trajectory estimation. Because the memory needs were acceptable, it was appropriate for resource-constrained robotic systems.

Finally, the SLAM system accurately calculated the robot's course and built a thorough picture of the area. Because of its capacity to manage loop closures, limit sensor noise, and deliver real-time performance, it is a dependable and efficient SLAM solution for various robotic applications.

Several areas for further development and improvement can be recognized

based on the findings of comparing the SLAM algorithms in different background situations:

1. **Robustness in Dynamic Environments:** Improving the robustness of SLAM algorithms in dynamic environments is one potential area for future investigation. This could include creating strategies to handle better-moving objects, such as dynamic object identification and tracking, to keep them from interfering with the mapping and localization processes. Furthermore, investigating ways for adaptive or online map updates to accommodate environmental changes might increase the algorithms' performance in dynamic circumstances.
2. **Occlusion Handling:** Another area for future development is the development of more effective occlusion handling systems. Occlusions, when objects or impediments obscure the sensors' line of sight, might present difficulties for SLAM algorithms. It would be beneficial to investigate ways for robustly predicting the robot's pose and sustaining accurate mapping even in the presence of occlusions. This could entail combining data from different sensor modalities, such as LiDAR and RGB-D cameras, to tackle occlusion-related concerns.
3. **Optimization of Real-Time Performance:** While the evaluated SLAM algorithms displayed real-time performance, more optimization can be explored to improve efficiency. This can include investigating parallel computing approaches, hardware acceleration with GPUs or specialized CPUs, or algorithmic optimizations to lower processing requirements while maintaining accuracy. Improving the algorithms' efficiency would allow them to be used on resource-constrained systems or in scenarios requiring even faster processing.
4. **Benchmarking and Comparison:** Standardised benchmarks and evaluation metrics help validate and compare the performance of SLAM algorithms in diverse contexts. Creating benchmark datasets such as [43] with variable background circumstances, ground truth data, and typical performance indicators will help academics to statistically analyze their advancements and improvements and facilitate fair comparisons across different algorithms.
5. **Use Deep Learning in SLAM:** As point clouds are not structured, it is impossible to use parallel processing like CUDA, so Deep Learning architectures are not typically preferred. Recently a CNN-type architecture for 3D

point clouds was proposed [44]. One can use such type of deep learning architecture for edge and corner detection to improve the results of SLAM further.

References

- [1] Point Clouds are Eating the World, One Application at a Time, <https://www.sigarch.org/point-clouds-are-eating-the-world>, Jan. 2021.
- [2] G. Roe, "The Zamani Project - Heritage Documentation Beyond the Point Cloud - LiDAR News," <https://lidarnews.com/articles/the-zamani-project-heritage-documentation-beyond-the-point-cloud>, jan. 2016.
- [3] Top Technologies Shaping Metaverse, <https://www.bisinfotech.com/top-technologies-shaping-metaverse>, Feb. 2022.
- [4] L. Pickup, "Perspectives on the future of the car – MIND-sets Knowledge Center," <https://mobilitybehaviour.eu/2017/08/07/perspectives-on-the-future-of-the-car/>, aug 7 2017.
- [5] Y. Bisheng, L. Fuxun, and H. Ronggang, "Progress, challenges and perspectives of 3D LiDAR point cloud processing," *Acta Geodaetica et Cartographica Sinica*, vol. 46, no. 10, pp.1509, 2017.
- [6] X. Huang, G. Mei, J. Zhang, and R. Abbas, "A comprehensive survey on point cloud registration," *arXiv preprint*, arXiv:2103.02690, 2021.
- [7] F. Remondino, "From point cloud to surface: the modeling and visualization problem," *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pp. 34, 2003.
- [8] X. Xie, L. Bai, and X. Huang, "Real-time LiDAR point cloud semantic segmentation for autonomous driving," *Electronics*, pp.11, vol. 1, no.11, 2021.
- [9] H. Wang, C. Wang, and L. Xie, "Lightweight 3-D localization and mapping for solid-state LiDAR," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1801–1807, 2021.
- [10] R. B. Rusu and S. Cousins, "3D is here: Point cloud library (PLC)," *In 2011 IEEE international conference on robotics and automation*, pp. 1–4, 2011.

- [11] W. Wang, D. Zhu, X. Wang, Y. Hu, Y. Qiu, C. Wang, Y. Hu, A. Kapoor, and S. Scherer, "Tartanair: A dataset to push the limits of visual SLAM," *In 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4909–4916, 2020.
- [12] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (fpfh) for 3D registration," *In 2009 IEEE international conference on robotics and automation*, pp. 3212–3217, 2009.
- [13] H. Hoppe, T. Deroose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized point clouds," 1992.
- [14] Y. Guo, M. Bennamoun, F. Sohel, M. Lu, J. Wan, and N. M. Kwok, "A comprehensive performance evaluation of 3D local feature descriptors," *International Journal of Computer Vision*, vol. 116, pp. 66–89, 2016.
- [15] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv preprint*, arXiv:1801.09847, 2018.
- [16] J. Behley and C. Stachniss, "Efficient surfel-based slam using 3D laser range data in urban environments," *In Robotics: Science and Systems*, pp. 59, 2018.
- [17] J. Zhang and S. Singh, "LOAM: LiDAR odometry and mapping in real-time," *In Robotics: Science and Systems*, vol. 2, pp. 1–9, Berkeley, CA, 2014.
- [18] A. Broggi, S. Cattani, M. Patander, M. Sabbatelli, and P. Zani, "A full-3D voxel-based dynamic obstacle detection for urban scenario using stereo vision," *In 16th International IEEE Conference on Intelligent Transportation Systems (ITSC2013)*, pp. 71–76, 2013.
- [19] D. Wang, M. Hollaus, E. Puttonen, and N. Pfeifer, "Fast and robust stem reconstruction in complex environments using terrestrial laser scanning," *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pp. 41, 2016.
- [20] P. Dellenbach, J.-E. Deschaud, B. Jacquet, and F. Goulette, "CT-ICP: Real-time elastic LiDAR odometry with loop closure," *In 2022 International Conference on Robotics and Automation (ICRA)*, pp. 5580–5586, 2022.
- [21] X.-F. Han, J. Jin, M.-J. Wang, W. Jiang, L. Gao, and L. Xiao, "A review of algorithms for filtering the 3D point cloud," *Signal Processing: Image Communication*, vol. 57, pp. 5, 2017.

- [22] F. Poux and R. Billen, "Voxel-based 3D point cloud semantic segmentation: Unsupervised geometric and relationship featuring vs deep learning methods," *ISPRS International Journal of Geo-Information*, vol. 8, no. 5, pp. 213, 2019.
- [23] E. Grilli, F. Menna, and F. Remondino, "A review of point clouds segmentation and classification algorithms," *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 42, pp. 339, 2017.
- [24] F. Pomerleau, F. Colas, R. Siegwart, et al., "A review of point cloud registration algorithms for mobile robotics," *Foundations and Trends in Robotics*, vol. 4, no. 1, no. 1–104, 2015.
- [25] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," *In Proceedings third international conference on 3-D digital imaging and modeling*, pp. 145–152, 2001.
- [26] D. Teijeiro, M. Amor, R. Doallo, and D. Deibe, "Interactive visualization of large point clouds using an autotuning multiresolution out-of-core strategy," *The Computer Journal*, pp. 12, 2022.
- [27] S. Agarwal, K. Mierle, and T. C. S. Team, Ceres Solver, 3 2022.
- [28] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly of applied mathematics*, vol. 2, no. 2. pp. 164–168, 1944.
- [29] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous robots*, vol. 34, pp.189–206, 2013.
- [30] G. Bradski. The OpenCV Library. Dr. Dobb's Journal of Software Tools, 2000.
- [31] S. Thrun, D. Koller, Z. Ghahramani, H. Durrant-Whyte, and A. Y. Ng, "Simultaneous mapping and localization with sparse extended information filters: Theory and initial results," Springer, 2004.
- [32] J.-A. Fernández-Madrigal, Simultaneous Localization and Mapping for Mobile Robots: Introduction and Methods: Introduction and Methods, *IGI global*, 2012.
- [33] T. A.-Q. Tawiah, "A review of algorithms and techniques for image-based recognition and inference in mobile robotic systems," *International Journal of Advanced Robotic Systems*, pp. 17, no. 6, pp. 1729881420972278, 2020.

- [34] M. Montemerlo and S. Thrun, *FastSLAM: A scalable method for the simultaneous localization and mapping problem in robotics*, 7th Edition, Springer, 2007.
- [35] D. Li, W. Yang, X. Shi, D. Guo, Q. Long, F. Qiao, and Q. Wei, "A visual-inertial localization method for unmanned aerial vehicle in underground tunnel dynamic environments," *IEEE Access*, vol. 8, pp. 76809–76822, 2020.
- [36] R. Eustice, M. Walter, and J. Leonard, "Sparse extended information filters: Insights into sparsification," *In 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3281–3288, 2005.
- [37] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based SLAM," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp.31–43, 2010.
- [38] Z. Xuexi, L. Guokun, F. Genping, X. Dongliang, and L. Shiliu, "SLAM algorithm analysis of mobile robot based on LiDAR," *In 2019 Chinese Control Conference (CCC)*, pp. 4739–4745, 2019.
- [39] S. Thrun and J. J. Leonard, *Simultaneous Localization and Mapping*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.
- [40] J. Neira and J. D. Tardós, "Data association in stochastic mapping using the joint compatibility test," *IEEE Transactions on robotics and automation*, vol. 17, no. 6, pp. 890–897, 2001.
- [41] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G2O: A general framework for graph optimization," *In 2011 IEEE International Conference on Robotics and Automation*, pp. 3607–3613, 2011.
- [42] C. Stachniss, G. Grisetti, and W. Burgard, "Information gain-based exploration using rao-blackwellized particle filters," *In Robotics: Science and systems*, vol. 2, pp. 65–72, 2005.
- [43] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," *In 2012 IEEE conference on computer vision and pattern recognition*, pp. 3354–3361, 2012.
- [44] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3D classification and segmentation," *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017.