POLITECNICO DI TORINO

Master of Science in Automotive Engineering

Master Thesis

Simultaneous Localization and Mapping algorithms benchmark for Autonomous Driving

Supervisor
Prof. Andrea Tonoli

Candidate
Stefano Vitrani

Company Advisors
Fabio Danna
Anna Boschi
Yannick Giovanakis

A.Y. 2023-2024

Contents

List of Tables

List of Figures

Abstract

In the context of Autonomous Driving (AD), Simultaneous Localization and Mapping (SLAM) algorithms are of fundamental importance in enabling vehicles to navigate and interact with the surrounding environment, with no input from a human driver. These algorithms are used to construct a map of the surroundings while estimating, at the same time, the position and trajectory of the vehicle inside of it.

This thesis, developed as a collaboration between **Politecnico di Torino** and **Italdesign Giugiaro**, aims to adapt some of the best open-source SLAM algorithm to an internal AD project of the company, the autonomous vehicle prototype called **TechDemo**.

The focus of this work is to integrate the algorithms in the software of the TechDemo to generate a map of the surroundings and conduct benchmark testing finalised at selecting the best one for the application.

Chapter 1

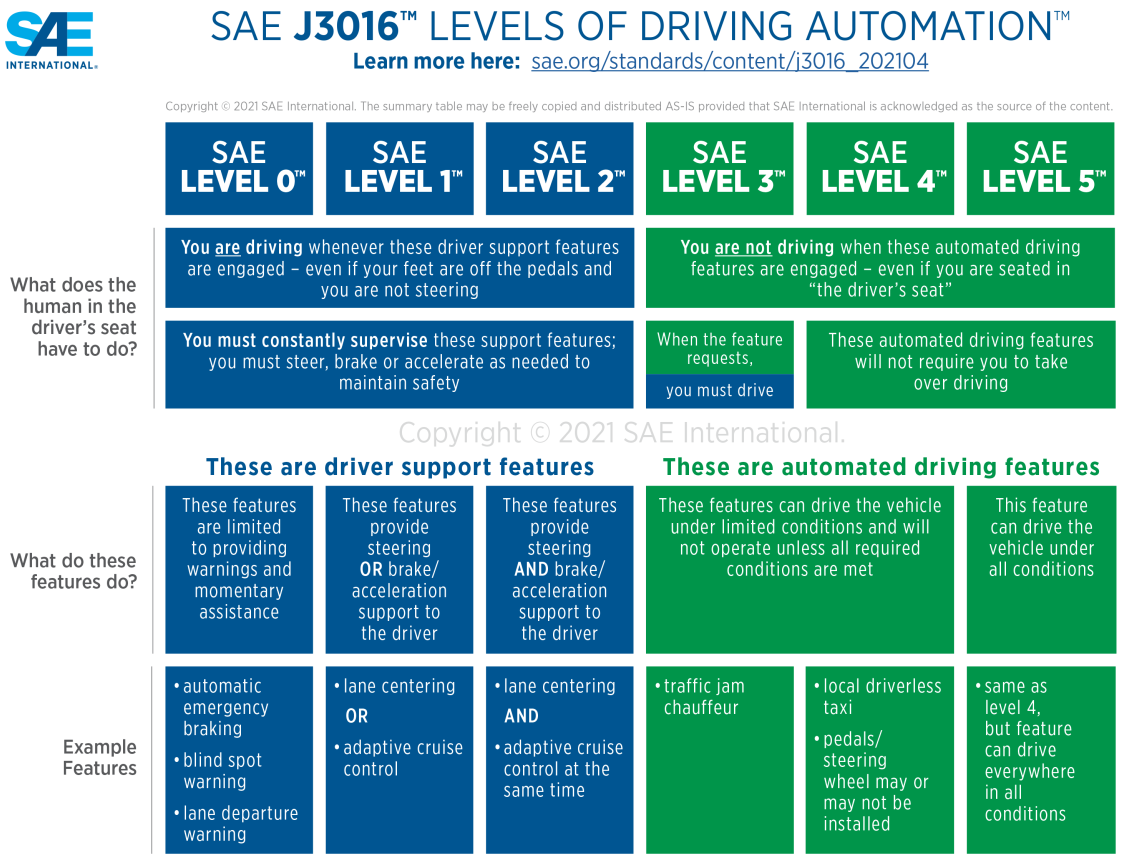
Introduction

Motorized transportation played a fundamental role in shaping the modern world, revolutionizing the transport system and allowing people and goods to travel vast distances in short time. From the advent of steam engines to the development of electric motors, the evolution of the transportation system has been characterized by innovation, progress and social impact. In recent years, the incredible development made in micro-computing and sensor advancement has shaped the way for a modern-day revolution: Autonomous Driving (AD).

1.1 Autonomous Driving

Autonomous Driving (AD) technology has emerged as a focal point of innovation within the automotive industry. Advanced Driving Assistance Systems (ADAS) and active safety systems are already present in the most recent commercially available vehicles, and since their introduction, they have played a pivotal role in reducing accidents and increasing road safety.

These systems represent the first step towards AD. However, fully autonomous vehicles are currently limited by today's technology, legal regulations, and social perception, making their launch in today's market unlikely. Nevertheless, advancements in Artificial Intelligence (AI), sensors, and computing power hold the promise of changing this situation in the next few decades.



SAE **J3016**™ LEVELS OF DRIVING AUTOMATION™

Learn more here: sae.org/standards/content/j3016_202104

	SAE LEVEL 0™	SAE LEVEL 1™	SAE LEVEL 2™	SAE LEVEL 3™	SAE LEVEL 4™	SAE LEVEL 5™
What does the human in the driver's seat have to do?	**You are driving** whenever these driver support features are engaged – even if your feet are off the pedals and you are not steering			**You are not driving** when these automated driving features are engaged – even if you are seated in "the driver's seat"		
	You must constantly supervise these support features; you must steer, brake or accelerate as needed to maintain safety			When the feature requests, you must drive	These automated driving features will not require you to take over driving	
	These are driver support features			**These are automated driving features**		
What do these features do?	These features are limited to providing warnings and momentary assistance	These features provide steering **OR** brake/ acceleration support to the driver	These features provide steering **AND** brake/ acceleration support to the driver	These features can drive the vehicle under limited conditions and will not operate unless all required conditions are met		This feature can drive the vehicle under all conditions
Example Features	• automatic emergency braking • blind spot warning • lane departure warning	• lane centering **OR** • adaptive cruise control	• lane centering **AND** • adaptive cruise control at the same time	• traffic jam chauffeur	• local driverless taxi • pedals/ steering wheel may or may not be installed	• same as level 4, but feature can drive everywhere in all conditions

Figure 1.1: SAE J3016 level of automation [1]

This promising progress in AD technology led the Society of Automotive Engineers (SAE) to draft the normative SAE J3016 [1], which defines six levels of automation, as reported in Figure 1.1 and briefly summarized below:

- **Level 0 - No automation**: The driver is fully responsible for driving operation. The vehicle may have driving assistance features like warnings, but they do not perform any driving tasks.
- **Level 1 - Driver Assistance**: The vehicle can assist with steering or acceleration/deceleration, but not simultaneously. Simple ADAS falls into this level. The driver is still fully responsible for driving operation.
- **Level 2 - Partial Automation**: The vehicle can assist with steering and acceleration/deceleration simultaneously under certain conditions. However, the

driver must remain engaged and monitor the systems and is fully responsible for driving operation.

- **Level 3 - Conditional Automation**: The vehicle can perform every driving operation under certain conditions and environments. The driver can disengage from driving operation but must be available to take control when prompted by the vehicle.
- **Level 4 - High Automation**: The vehicle can fully operate driving tasks, and the driver is not required, but only in controlled environments. The system can handle most situations but may require human intervention if beyond its Operational Design Domain (ODD).
- **Level 5 - Full Automation**: The vehicle can operate without human intervention in all conditions and every environment. Manual driving controls are not required, and it can be operated without a human driver.

1.2 Thesis outline

In the context of AD, localization and mapping algorithms play a pivotal role in enabling level 3 and up of automation. The aim of this thesis is to analyze and integrate some of the best and available open-source algorithms to the AD project of the company.

The work is structured as follows:

- **Chapter 2** introduces the company's AD prototype, explaining its systems and architecture.
- **Chapter 3** presents the Simultaneous Localization and Mapping (SLAM) algorithms analyzed, reporting their features and general functioning.
- **Chapter 4** explains the application in which the algorithms will be integrated.
- **Chapter 5** presents the testing procedure and reports the results comparing them.
- **Chapter 6** is the final chapter, which contains a summary of this thesis and presents future work.

Chapter 2

System architecture

The autonomous prototype *TechDemo* was developed and built internally in 2018 by the original team at Italdesign as the company's first SAE Level 4 (section 1.1) AD project. The project was then suspended until mid-2022, when a new and smaller team resumed its development. However, some of the original documentation was missing, and the new team has since made significant efforts to reverse-engineer the prototype and develop new and improved systems and software.

Therefore, the first objective of this work was to achieve a comprehensive understanding of the TechDemo: its construction, architecture and operational systems.

In what follows, a complete description of the prototype's systems will be presented.

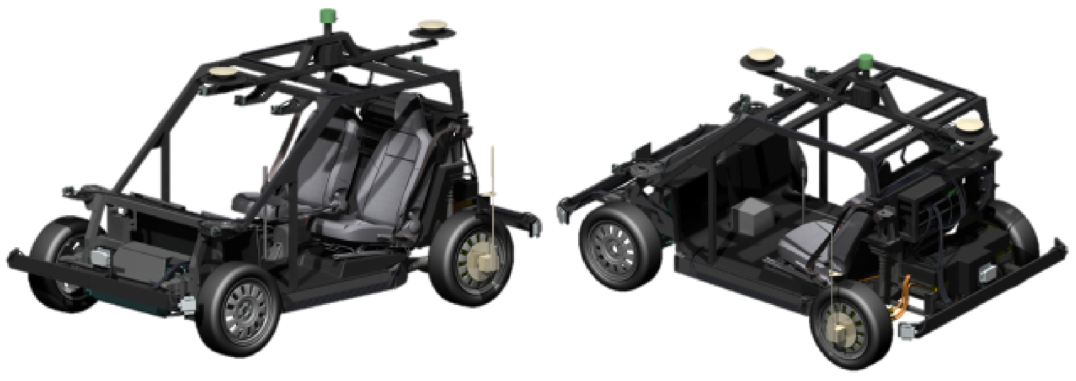


Figure 2.1: TechDemo rendering

2.1 Hardware

The vehicle frame utilized for the prototype is based on a Volkswagen UP!, which has been modified to accommodate all the essential components required for AD. The architecture description will primarily concentrate on the AD-related components. A rendering of the vehicle is depicted in Figure 2.1.

2.1.1 Processing Units

The prototype integrates two main processing units: a **dSPACE MicroAutoBox II** (Figure 2.2) and an **NVIDIA Jetson Orin** (Figure 2.3).

The **dSPACE MicroAutoBox II** is a real-time embedded system designed for control and prototyping in automotive development. A complete model of the vehicle can be loaded into its memory and utilized as the interface between the actuation commands (steering input, torque input, brake input) and the actuators. It serves the same purpose as a conventional Electronic Control Unit (ECU), with the advantage of being easily re-programmable according to the system's requirements. Moreover, it is capable of communicating using various network protocols, thereby facilitating message exchange between different types of devices.

In our system, the MicroAutoBox II communicates via Controller Area Network (CAN) with the vehicle control network, enabling access to diagnostic messages (Battery Management System (BMS) data, steering position, etc.) and sending actuation commands to the motors, braking and steering ECUs. Additionally, it communicates through Ethernet via User Datagram Protocol (UDP) with the other processing unit, sending diagnostic and informational messages and receiving actuation requests.

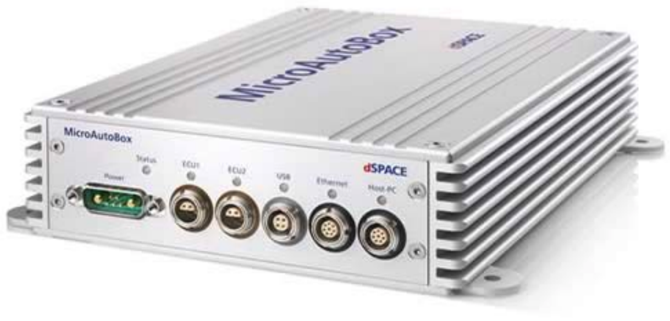


Figure 2.2: dSPACE MicroAutoBox II

The second processing unit is the **NVIDIA Jetson Orin** shown in Figure 2.3. It is an AI platform designed for autonomous machines and edge computing. It integrates multiple Central Processing Unit (CPU)s and Graphics Processing Unit (GPU)s, along with dedicated AI and computer vision hardware accelerators, delivering outstanding processing power while still being optimized for energy efficiency. Its exceptional capabilities and advanced features make it the perfect candidate for perception, path-planning, and decision-making processing unit.

In our system, the NVIDIA Jetson Orin serves as the on-board computer where every AD-related feature is computed. This is where all the data from the sensors is gathered and processed to gain knowledge of the surroundings, including obstacle and object recognition through AI models, trajectory planning, and state estimation. This unit is then responsible for sending suitable acceleration, braking, and steering requests to the dSPACE MicroAutoBox II.

Figure 2.3: NVIDIA Jetson Orin

2.1.2 Sensors

The TechDemo is equipped with various sensors that allow it to perceive its surroundings and navigate safely. These sensors provide essential data about the surrounding environment and are used to detect the presence of other vehicles, people, road signs, traffic lights, lane markings, etc. Some of the sensors are also utilized by the SLAM algorithms to map the environment and estimate the vehicle's pose.

Figure 2.4 provides an overview of the sensors on board of the vehicle.

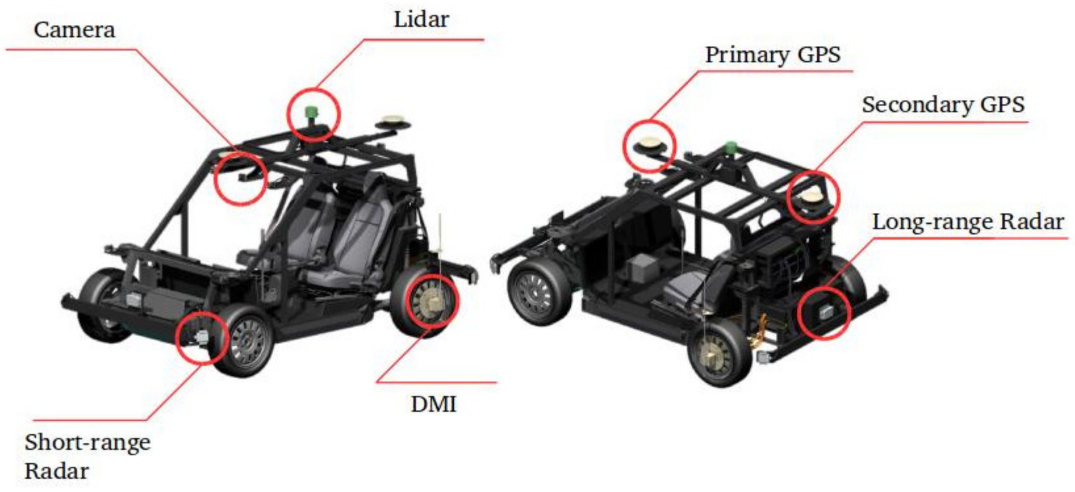


Figure 2.4: TechDemo sensors overview

- **Light Detection And Ranging (LiDAR)**

  This is the primary sensor used for localization. Lidar sensors emit laser beams and use the reflection time to measure the distance of surrounding objects from the sensor, providing a cloud of points used to reconstruct a 3D map of the environment, as shown in Figure 2.6.

  In our system, the **Velodyne VLP-16** lidar shown in Figure 2.5 is used. This sensor is a 16-channel 360° horizontal field 30° vertical field lidar capable of delivering up to 300,000 points per second, translating to a complete 360° environmental scan at a rate of 10 Hz.

  This sensor is used for mapping, state estimation, and object detection.

- **Camera**

  The TechDemo integrates several cameras. The primary one is the **Intel RealSense D435** stereo-camera shown in Figure 2.7, which is a Full-HD combined depth and RGB camera delivering a video feed of up to 90 fps.

  The camera is used to capture visual data about the surroundings of the vehicle, which is used by computer vision algorithms and AI models to detect and

Figure 2.5: Velodyne VLP-16 Lidar sensor

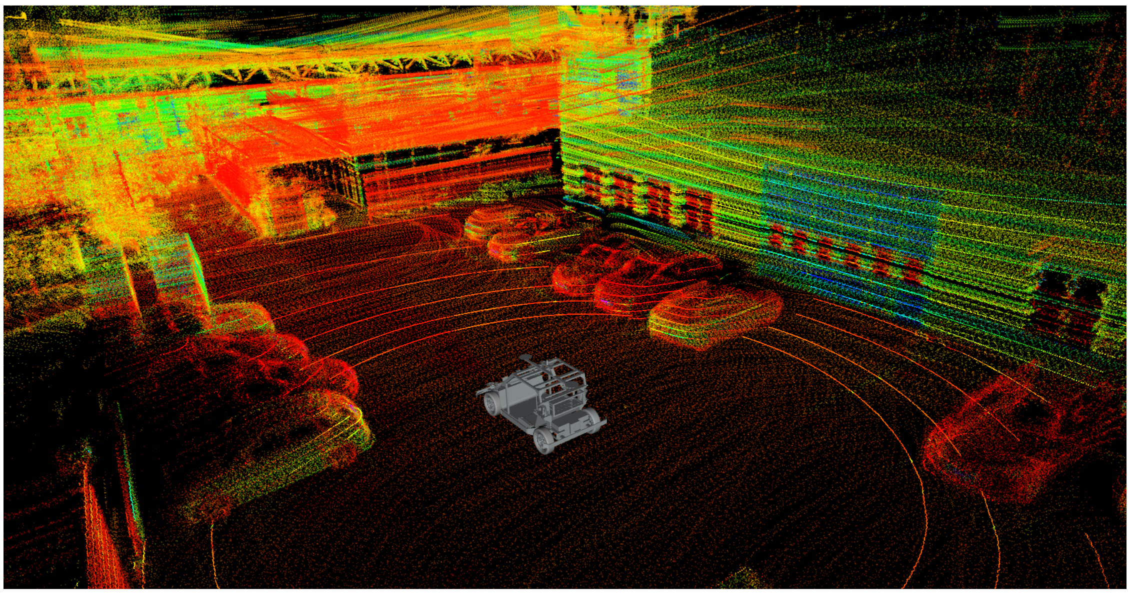


Figure 2.6: Point cloud map generated through Velodyne VLP-16

recognize objects, obstacles, vehicles, traffic lights, road signs, and pedestrians. Figure 2.8 shows an example of object detection through the camera.

- **Global Positioning System (GPS)**

  Global Positioning System (GPS) receivers provide a global and absolute measure of the vehicle's position. Although not sufficient by itself due to its preciseness, it can be used in combination with other sensors.

  The GPS system in use is part of the complete solution **Applanix LV POS**

Figure 2.7: Intel RealSense D435 camera

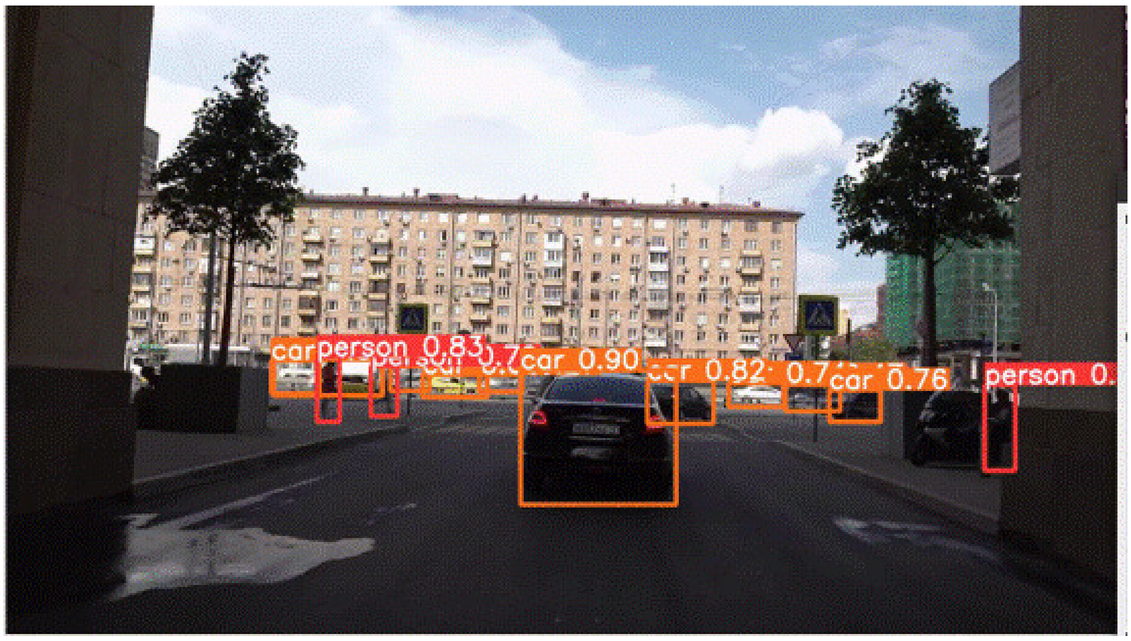


Figure 2.8: Object Detection through camera

420 shown in Figure 2.9, which also includes an Inertia Measurement Unit (IMU) and Distance Measurement Unit (DMI) sensor. This sensor is primarily used for pose initialization when combined with a geo-referenced map. It is also used in SLAM algorithms to correct the state estimation error.

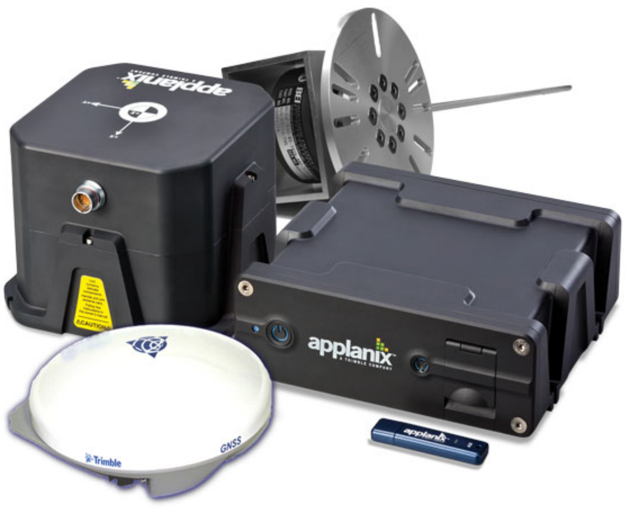


Figure 2.9: Applanix LV POS 420 system

- **Inertia Measurement Unit (IMU)**

  IMU sensors provide information about the vehicle's acceleration, angular rate, and, in some models, orientation.

  The sensor in use is a 9-Axis IMU integrated into the Applanix system shown in Figure 2.9, capable of 3-Axis acceleration, 3-Axis angular rate, and 3-Axis orientation measurements. The Applanix Position and Orientation System (POS) solution also integrates the POS Computer System (PCS), which is a computational unit that processes the data coming from the related sensors and delivers it to the AD processing unit via Ethernet. The data obtained this way is filtered and more reliable.

  The data is used in SLAM and sensor fusion algorithms for navigation, planning, and state estimation.

- **Distance Measurement Unit (DMI)**

  The DMI is a wheel sensor used to estimate vehicle speed and traveled distance.

It is included in the Applanix system and is used in combination with IMU and GPS sensors by the PCS, but is not directly used in any AD algorithm.

The TechDemo includes other sensors, such as long-range and short-range Radio Detection And Ranging (RADAR), which are currently not utilized.

2.1.3 Others

Other hardware components not strictly related to AD include a **Netgear Ethernet Switch**, used to connect and exchange data between the sensors and the processing units, and a **Power Supply Unit (PSU)**, which delivers power to the low-voltage components, providing the constant voltage required by each component to operate.

2.2 Software

Software plays a critical role in achieving autonomous driving capabilities, enabling vehicles to make decisions based on the surroundings and navigate safetly. In this section, the software used in the TechDemo will be described.

2.2.1 dSPACE Software

The dSPACE software was developed as a Master's thesis at Politecnico di Torino by the author of [5]. This software is responsible for receiving messages from various inputs and controlling the vehicle systems by sending commands via CAN. It is implemented as a MATLAB/Simulink model, which is then converted to C-Code, compiled, and uploaded to the memory by a dSPACE proprietary application. For a detailed explanation of the software's functioning, refer to [5].

Autonomous Driving software

The AD computer, NVIDIA Jetson Orin, runs a Linux-based Operating System (OS), which is an NVIDIA-optimized version of Canonical's Ubuntu 22.04. On this platform, various software components are utilized, as described below.

2.2.2 Robot Operating System (ROS)

The Robot Operating System (ROS) is an open-source framework specifically designed for robot applications, including autonomous driving vehicles. It provides a set of Python and C++ software libraries and tools aimed at simplifying the development of complex robotic systems.

One of the key features of ROS is its publish-subscribe messaging system, which facilitates communication between different hardware actors (such as sensors and processing units) and software modules. This enables seamless integration and coordination of distinct hardware and software components, allowing for a completely modular design. Developers can select which components to use at each stage of development, facilitating testing of different modules without interfering with the functioning of the system.

Additionally, ROS includes a set of tools aimed at development and debugging, providing valuable support to developers and allowing them to focus on higher-level features.

ROS is available in two versions, each with its own distributions. The legacy version, referred to as ROS1, is nearing its End-of-Life (EOL) in May 2025. However, the new and improved version, ROS2, is becoming the standard for future robot applications.

Although it is recommended to develop for ROS2, the SLAM algorithms discussed in chapter 3 and used for offline-map generation are all ROS1-based. This decision was made because these algorithms have been in development for a longer time, and ROS2-based alternatives are not yet as mature. On the other hand, the software running on the AD processing unit of the TechDemo is based on ROS2, as it is expected to become the standard for future robot applications.

2.2.3 Sensors drivers

The measurements coming from the sensors are delivered as a stream of binary data through different network protocols. As a result, software interfaces, known as drivers, are necessary to facilitate communication between the sensors and the processing unit.

All the AD algorithms are developed within the ROS framework, so the drivers must be designed to work seamlessly with ROS. Typically, manufacturers provide ROS drivers directly for their sensors, or community-developed drivers are available for widely used sensors. This is the case for every sensor used in the TechDemo, except for the Applanix LV POS 420 driver, which lacked a ROS2 version. Consequently, the driver was developed internally based on the ROS1 version to ensure compatibility with the TechDemo's ROS-based environment.

2.2.4 Autoware

At the core of the TechDemo's AD capabilities is Autoware [6]. Autoware is an open-source platform based on ROS2, developed by the Autoware Foundation, with the goal of democratizing AD technology by making it accessible to developers and researchers. To achieve this, Autoware provides a comprehensive set of modules and tools, including algorithms, libraries, and frameworks, enabling developers to efficiently build AD systems.

A notable feature of Autoware is its modularity, which facilitates the development of autonomous vehicles by breaking down functions into dedicated and interchangeable components. This approach allows for efficient testing and development.

Autoware's architecture consists of two main modules: the **Core** module and the **Universe** module. Each module provides components that are designed to be reusable and interchangeable.

Autoware Core

The Core module contains basic components that provide the fundamental functionalities required by AD systems. These components are developed and maintained directly by the Autoware Foundation and serve as a solid foundation for building any autonomous application.

Autoware Universe

The Universe module contains components that extend the functionality and capability of the Core module. While the Autoware Foundation hosts a wide variety of Universe components and supervises their quality control, it does not directly develop them. These components may be provided, for example, by sensor or hardware manufacturers to implement their products in the Autoware framework.

Developers also have the option to design their own Autoware Universe components to integrate, enhance, or even replace Core components, depending on their specific requirements.

A qualitative overview of the Core/Universe concept is depicted in Figure 2.10.

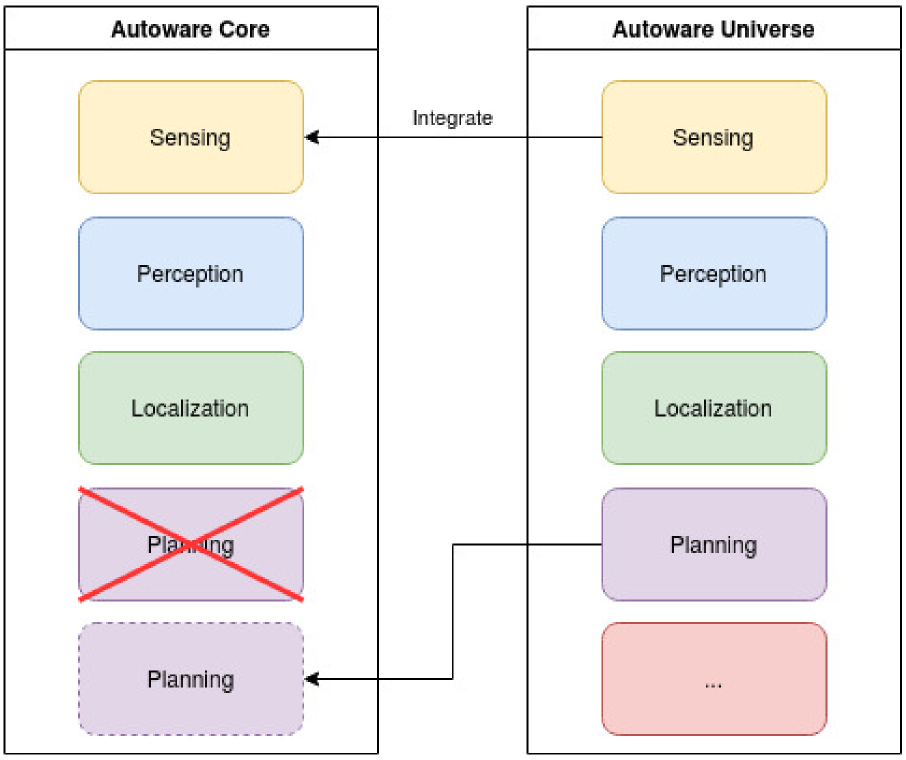


Figure 2.10: Autoware Core/Universe architecture

Chapter 3

SLAM Algorithms

In this chapter, the features and workings of each algorithm will be explained. The following algorithms are analyzed and utilized in this work:

- Lidar-Inertial Odometry via Smoothing and Mapping (LIO-SAM)
- Lightweight and Ground Optimized Lidar Odometry and Mapping (LeGO-LOAM)
- HDL Graph SLAM
- Fast Lidar-Inertial Odometry (FAST-LIO)

Since many of the algorithms presented here are based on the so-called **graph-based SLAM**,a brief introduction of this concept will be given before describing the algorithms.

3.1 Introduction to graph-based SLAM

Graph-based methods have emerged as one of the most prevalent approaches for fulfilling SLAM challenges. In this method, the problem is formulated as a graph, where the nodes represent the poses of the system at various points in time, and the edges represent the relationships or constraints between these poses [7]. These constraints, often referred to as factors, provide information about the spatial relationships between poses, such as relative distances or angles.

The construction of the map in a graph-based SLAM system involves finding the spatial configuration of the nodes that best satisfies all the constraints, optimizing an objective function that balances the alignment of the poses with the imposed constraints. These constraints can originate from measurements acquired by sensors onboard the robot, or vehicle, such as lidars, cameras, or inertial measurement units (IMUs). However, they can also be derived from the internal dynamics of the system, for instance, by detecting loop closures when the robot revisits a previously visited location (3.2.2).

To illustrate the concept of graph-based SLAM, consider the architecture depicted in Figure 3.3. In this illustration, each node in the graph represents the state of the robot at a specific time instance, depicted in blue. The edges between nodes represent the constraints derived from sensor measurements or odometry information. Through iterative optimization processes, the system refines the positions of the nodes to minimize the discrepancies between observed measurements and predicted poses.

3.2 LIO-SAM

LIO-SAM is an efficient tightly-coupled lidar-inertial odometry framework built atop a factor graph, developed by the authors of [2] for robot applications, which take advantage of a lidar and an IMU sensors, and optionally of a GPS antenna.

3.2.1 Features

Sensor fusion

The framework exploits raw IMU measurements to estimate the sensor motion during a lidar scan and use this information for point cloud deskew. The obtained lidar odometry is then used to estimate the bias of the IMU.

In addition to *local* IMU and lidar measurements, the algorithm is able to incorporate *global* measurements, such as GPS positioning and compass heading, if and when they are available.

Feature extraction and Keyframe selection

Lidar odometry is usually performed by comparing two subsequent scan frames via scan-matching algorithm and finding the relative transformation between the two. However, comparing two full point clouds can be computationally inefficient and highly time consuming. To solve this problem, a popular approach is using feature-based matching methods. LIO-SAM adopts this method: instead of directly compare two subsequent lidar scan, whenever a new scan frame is ready, it performs the so called *feature extraction*.

Edge and planar features are extracted by evaluating the roughness of points over a local region: points with high roughness are classified as edge features, while points with small roughness are classified as planar features. [2]

With the objective of reducing the computational cost of the algorithm, LIO-SAM exploits also the concept of *keyframe* selection. Processing every lidar scan frame which, with our sensor, are delivered at 10 Hz is indeed unfeasable. For that reason, the algorithm select the next lidar frame F_{i+1} to be processed only when the change in the robot pose x_{i+1} exceed a certain user-defined threshold if compared to the previous state x_i. All the other frames within two keyframes are discarded.

Loop closure

A loop closure feature is also present, which allows for correction of the trajectory when the robot reaches a spot previously mapped. This is particularly useful when a GPS is not available and particularly effective in correcting the trajectory altitude drift.

Figure 3.1 shows an example of the loop closure feature.

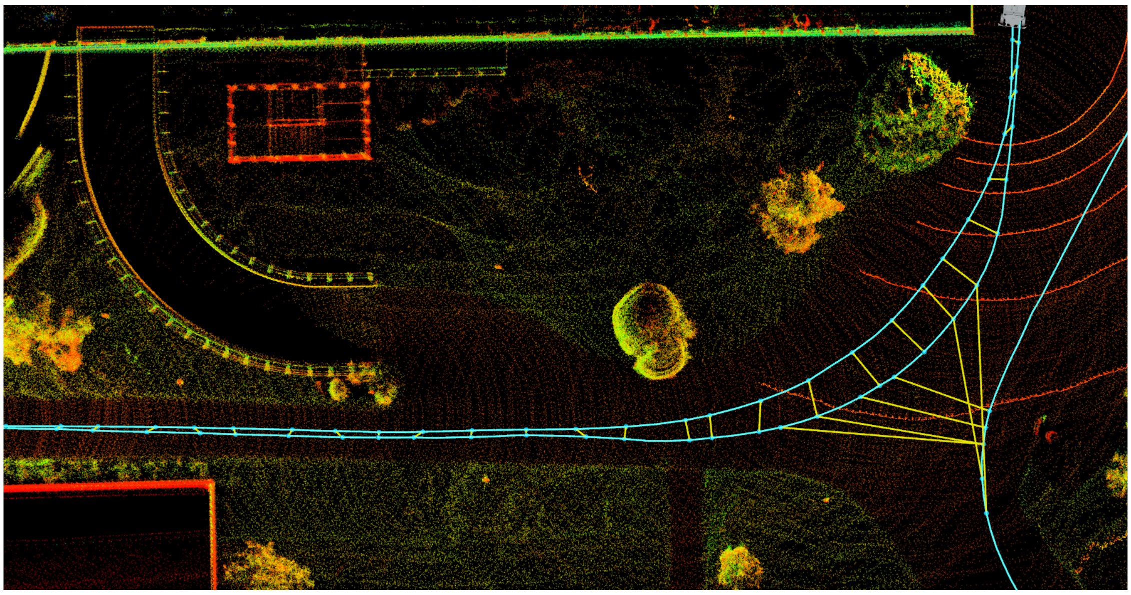


Figure 3.1: LIO-SAM trajectory loop closure detail

Real time application

Due to the reasons described in 3.2.1, this algorithm is particularly computationally efficient, which makes it an optimal candidate for online and real time usage, i.e. using the algorithm to allow an autonomous vehicle to drive in an unexplored before area.

Although this use case is not the objective of neither Autoware nor the TechDemo, it certainly is a big advantage for future development.

3.2.2 Working principle

LIO-SAM seeks to estimate the state and the trajectory of a system by observation of data coming from the sensors (Lidar, IMU and, optionally, GPS). The problem can be formulated as a Maximum a posteriori (MAP) problem [2]. This problem is modelled as a factor graph with four factors and one variable (the state x_i) attributed to the nodes of the graph.

An overview of the factor graph is presented in Figure 3.2, taken directly from the original paper [2].

In what follows, a brief description of the factors is given. For a more detailed description and mathematical formulations, refer to the original LIO-SAM paper [2].

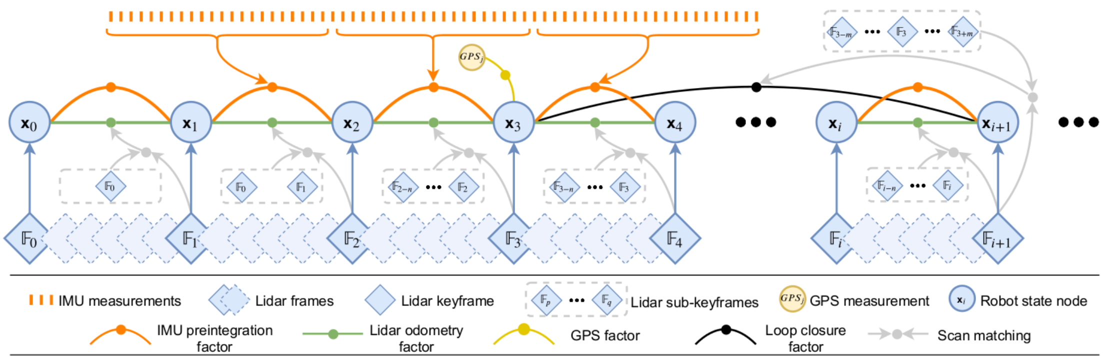


Figure 3.2: Factor Graph Overview [2]

IMU Preintegration factor

The first factor is obtained from processing the raw data delivered by the IMU. From the raw measurement of the IMU, it is possible to calculate the angular velocity and the acceleration of the system. A simple integration of the measurement over Δt is performed, thus obtaining the velocity, position and rotation of the system at time $t + \Delta t$. The authors apply then the IMU preintegration method explained in [8] to obtain the relative motion between two consecutive time steps.

The result obtained by IMU preintegration is the first constraint of the factor graph, i.e. IMU preintegration factor.

Lidar Odometry factor

This factor is obtained by processing data coming from the lidar.

Whenever a new scan is delivered from the lidar, the procedure described in 3.2.1 is performed: if the frame is considered a *keyframe* is kept and *feature extraction* [9] is obtained. The lidar odometry factor is then derived from the keyframe through three steps:

1. Sub-keyframes for voxel map

   Instead of using just the last keyframe for estimation, the first user-defined n most recent keyframes are used. This set of n keyframes is called sub-keyframes. The set of sub-keyframes is merged into a single voxel map M_i, which is composed of two voxel sub-maps, each containing the points classified as *edges* or *surfaces*. Then the voxel map M_i is downsampled as to avoid duplicates of the same feature falling in the same voxel cell (the dimensions of the voxel cell are user-defined).

2. Scan-matching

   This is where the feature based comparison between a new keyframe F_{i+1} and the past voxel map M_i is performed. The scan-matching algorithm used by LIO-SAM is described in [9]

3. Relative transformations

   Once each feature has been matched against its correspondent in the voxel map, the distance between the new pose and the last one can be computed. The transformation between the state x_i and the next x_{i+1} can be found by solving the optimization problem using the Gauss-Newton method.

   The relative transformation between a state x_i and the next state x_{i+1} is the lidar odometry factor.

GPS factor

Up to now, the pose and trajectory of the system has been obtained through integration of local measurements coming from lidar and IMU sensors. This means that if the measurements present errors, and in principle they always do, even the errors are integrated, leading to drift of the estimated trajectory with respect to the actual one. By adding a global measurement factor, i.e. the GPS factor, the trajectory can be corrected and aligned to the real one.

However, since the drift of a lidar-inertial odometry grows really slowly, it is not always necessary to use the GPS factor and, in some cases, it can be even avoided. This is useful because the computational cost of incorporating the GPS factor in the graph is not negligible.

Nevertheless, using GPS data in the algorithm can be an important benefit since it allows a precise orientation of the generated point cloud map in the global world frame.

Loop closure factor

Whenever a new state x_{i+1} is added to the factor graph, LIO-SAM first searches the graph for prior states that are close, in global coordinates, to it. If a candidate is found, compares the new frame F_{i+1} with a sub-keyframes set of the neighbours of the candidate's corresponding keyframe $F_{c\pm m}$, where m is a user defined parameter.

Then, similarly to 3.2.2, the relative transformation between x_{i+1} and the candidate state x_c is added to the graph.

This factor is particularly useful when no GPS factor is used or to correct the trajectory altitude when the GPS altitude readings are bad.

3.3 LeGO-LOAM

LeGO-LOAM [3] is a lightweight and ground-optimized SLAM algorithm developed by the same authors as LIO-SAM [2]. Its primary objective is computational cost optimization, enabling real-time operation on low-power embedded systems. It utilizes a lidar sensor and an IMU sensor. However, unlike LIO-SAM, the presence of the IMU sensor is optional.

3.3.1 Features

Feature extraction

Similar to LIO-SAM (3.2.1), a *feature extraction* algorithm is used to classify points from a point cloud in edge and planar features based on the roughness of said points. This allows to reduce the intensity of the computation.

Ground optimization

LeGO-LOAM relies on the presence of a ground plane for segmentation and optimization. This is particularly useful in the segmentation step to filter noise.

Loop closure

LeGO-LOAM integrates the ability to perform loop closure, in a similar way of LIO-SAM, allowing to correct the trajectory and minimize the drift, even in absence of a global measurement provider, such as GPS.

Real time application

Since LeGO-LOAM is built to be lightweight, that makes it particularly suitable for real time applications. Again, this goes beyond the scope of our application, but allows for future development.

3.3.2 Working principle

LeGO-LOAM aims at providing a lightweight system able to output a 6 Degrees of Freedom (DOF) pose estimation, as well as a point cloud of the mapped environment, receiving input data from a lidar sensor. To achieve this, the system is divided into five modules which are reported in Figure 3.3. In what follows, a brief explanation of the modules and how they work together is reported. For detailed description and mathematical formulation, refer to the original paper [3].

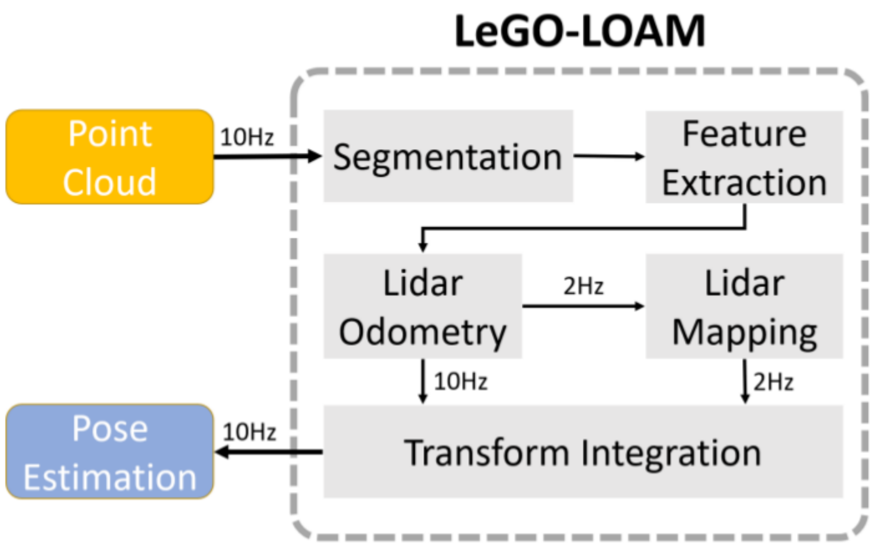


Figure 3.3: LeGO-LOAM system overview [3]

Segmentation module

The first step is performed by the segmentation module. This module takes a scan from the lidar as input and projects the 3D point cloud into a 2D range image. Each pixel in the range image represents a unique point in the point cloud and to each pixel in the image a value r_i is associated, which represents the distance of the corresponding point from the sensor. The first operation done on the obtained image is the *ground plane estimation*, which is performed as a column-wise evaluation [3]. Points obtained in this way are labeled as ground points and are not used for further segmentation.

Then, the proper segmentation is performed, used the method proposed in [10]. Points in the range image are now organized in clusters, that can be used to filter noise or small, movable and unreliable from a pose estimation point of view, objects, e.g. leaves, and use only fixed and reliable objects, e.g. buildings or trees, to perform the pose estimation. This method is extremely efficient as it greatly reduces the computational effort of point cloud processing.

Feature extraction module

The filtered image is then passed to the feature extraction module, exploiting the method proposed in [9]. The feature extraction algorithm is, however, applied to the segmented points and ground points, instead of the whole point cloud. At the end of the procedure, the image will be divided in sub-images with edges and planar features. LeGO-LOAM further reduce the computational cost of the next steps by selecting a user-defined number of edge features, which **do not** belong to the ground, and planar feature, which **must** belong to the ground.

Lidar odometry module

In this module two consecutive lidar scans (after processing) are compared through the scan-matching algorithm described in [9], seeking to estimate the transformation between a pose of the sensor and the previous one.

The algorithm compares the two scans and searches for two corresponding features in each scan. However, in view of computational efficiency a few changes have been made by the authors of [3] with respect to [9]:

1. Label matching

   Remembering that at this step two *segmented* images are being compared, to each feature is also associated a certain label. Therefore, the search for a feature's correspondent in the next scan is performed only on features that shares the same label. Aside improving performance, this method also helps with feature matching accuracy.

2. Two-steps L-M optimization

   LeGO-LOAM applies the two-steps Levenberg-Marquardt (L-M) method [3], which consists on dividing the 6 DOF in two sets of 3 DOF, using the planar features to obtain the transformation of the first set $[t_z, \theta_{roll}, \theta_{pitch}]$ and the edge features for the remaining DOF $[t_x, t_y, \theta_{yaw}]$

Lidar mapping module

The lidar mapping module is used as aid to further refine the pose estimation. Here the features are compared through a scan-matching algorithm, similar to 3.3.2, to a pre-saved whole point cloud map of the surrounding. The map is obtained by saving a point cloud from all previous feature sets. This procedure is certainly less computationally efficient, however the whole performance of the system is maintained since this operation is performed with low frequency.

Transform integration module

The last module fuses the pose estimation results coming from the lidar odometry and lidar mapping modules and provides the final pose estimation.

3.4 HDL Graph SLAM

HDL Graph SLAM is an offline SLAM graph optimization-based algorithm developed by the author of [11] as a component of a bigger suite that seeks to recognize and track people inside a point cloud. Differently from the other algorithms seen so far, it is designed not to be used as a real time pose estimator, but to generate an offline map to be used as ground truth in the other modules of the suite.

HDL refers to model of the lidar sensor used for testing by the original authors, a Velodyne HDL32e.

3.4.1 Features

Sensor fusion

HDL Graph SLAM only need a lidar sensor to function properly. However, it supports usage of other sensors such as IMU and GPS. Nevertheless, the IMU sensor is used only to rotate the pose aligning the gravity vector, which is useful to correct orientation error, but it is not used as a motion estimation constraint. The user can decide which sensor to utilize based on the needs, giving this algorithm an excellent flexibility and thus it is greatly system-agnostic.

Loop closure

Similar to LIO-SAM and LeGO-LOAM, this algorithm implements a loop closure feature, which is especially useful as trajectory correction when no other global measurements are available.

IMU orientation constraint

If an IMU sensor with embedded magnetic orientation sensor (i.e. 9-axis IMU sensor) is available, HDL Graph SLAM can take advantage of it, by using the data as graph constraint concerning 3D rotation. This is useful, in conjunction to loop closure, to reduce and correct trajectory drift.

Floor detection

The algorithm integrates a floor detection feature, which is used as constraint in the graph, assuring that each pose node have the same oriented floor. This is effective to reduce the orientation error, especially in large flat environment. This is used in conjunction to loop closure constraint to correct the pose estimation when no other measurement is available.

People detection

As specified before, this algorithm is a module of a bigger suite, designed with people detection in mind. This is worth of mention, since it can be really useful for future applications.

3.4.2 Working principle

The algorithm uses the graph method to solve the SLAM problem. In the graph, each node represents the parameters to be estimated, which in this case are sensor poses and landmark positions, and the edges are the constraints, mainly relative positions between the nodes.

In what follows, a brief description on how the constraints are constructed is presented. For a more detailed description and mathematical formulation, refer to [11].

Odometry constraint

The odometry constraint is obtained processing data coming from the lidar sensor. The sensor trajectory is estimated by comparing two consequent lidar scan frames through the Normal Distribution Transform (NDT) scan-matching algorithm described in [12]. Differently from other algorithms, no feature extraction optimization is performed and the whole point cloud is compared. However, this is by choice since the algorithm is designed for offline use and not real time applications which demand a certain efficiency.

Loop Closure constraint

In the next step, an algorithm is used to detect loops in the trajectory that can be added as graph edge to correct the accumulated error of the odometry constraint. The algorithm is similar to the one described in 3.2.2: first, loop candidates are detected by searching the graph for nodes which are closed, in the Euclidian space, to the newly pose. Then, the NDT scan-matching algorithm is applied between the lidar scan associated to the two nodes and, if a correspondance is found and is below a user-defined threshold, the loop is added to the graph as constraint. Each time a loop closure constraint is added to the graph, the graph is updated such to minimize the objective function (i.e. minimize the estimation error). [11].

Floor Detection constraint

This constraint is added to compensate the bend of the generated map due to the accumulated rotation error of the scan-matching algorithm. This is particularly effective in case of large flat environments, e.g. indoor applications. Every 10 seconds, the Random Sample Consensus (RANSAC) algorithm [13] is applied to the incoming lidar scan and, if the normal vector to the detected plane is under a certain threshold, i.e. the normal is vertical, the plane is classified as floor. The floor associated to the new scan is then compared to a fixed horizontal plane and the error between the two is added to the graph. Again, the graph is updated in order to minimize the objective function.

GPS constraint

If the ground is not flat, the constraint introduced in 3.4.2 is not effective, so a global measurement constraint, i.e. GPS factor, can be used to compensate for the estimation error. Whenever a GPS fix is available, it is associated (prior Universal Transverse Mercator (UTM) coordinates to Cartesian coordinates transformation) to the pose which is the closest in time. The error between the estimated, local pose and the global coordinates is then added as edge to the graph. Since in principle there is no way to associate GPS data to a pose, except from timestamp, it is crucial that GPS and lidar scans are synchronized, either by hardware or software means.

3.5 FAST-LIO

FAST-LIO [14] is a lidar-inertial odometry framework developed with computational efficiency and robustness in mind. It is originally designed to work in embedded systems operating in fast-motion and noisy environments, providing robust navigation by fusing lidar and IMU data through a tightly-coupled iterated extended Kalman filter [14].

3.5.1 Features

Feature extraction

Similar to 3.2 and 3.3, FAST-LIO integrates a feature extraction algorithm to decrease the processing computational effort. However, it can be disabled to allow direct odometry on raw lidar scans, achieving better mapping accuracy.

Online lidar-IMU calibration

FAST-LIO integrates a calibration algorithm to online estimate the extrinsics matrix between lidar and IMU frames. This could be really useful when this parameter is not known, as calculating it could be time consuming.

Real-time application

Thanks to it's lightweight, FAST-LIO is particularly suitable for real-time applications, even in case of low-power embedded systems, like robots or drones.

3.5.2 Working principle

In this section the pipeline that is the core of the algorithm is briefly explained. For detailed explanations and mathematical formulations refer to [14]. An overview of the system is showed in Figure 3.4.

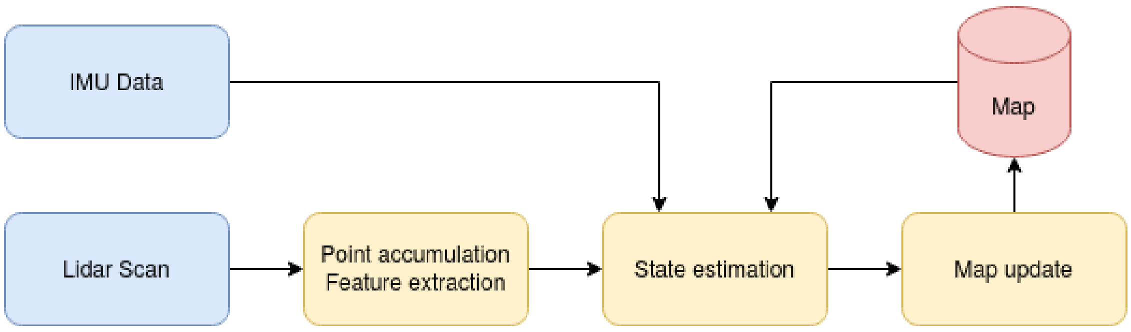


Figure 3.4: FAST-LIO system overview

Points accumulation and Feature extraction

The sequentially sampled lidar points are accumulated for a variable period of time, user-defined, to complete a scan. For 360 degree lidar, the period of accumulation should be the inverse of the spinning frequency, to make the scan concede with a complete revolution. If the option is enabled, at this step the feature extraction is performed, finding edge and planar features in the scan.

State estimation

To estimate the states, an iterated extended Kalman filter is used [14]. Whenever IMU data is received, forward propagation is performed and a predicted state is saved. Then, before fusing the propagated state with the lidar scan, backward propagation is performed to compensate for motion [14]. The state is then updated by calculating and fuse the residual with the predicted state following the algorithm described in [14].

Map update

Using the state update calculated in the state estimation module, each point in the scan is projected and transformed in the global frame. These points are then appended to the saved map containing points from all previous scans.

Chapter 4

Usecase description

The biggest challenge of this work was to develop software in order to integrate each algorithm to our system. The core code is open-source and can be found in each algorithm's paper, but it was modified and adapted to meet our specific needs.

In this chapter, the adaptation process will be described, showing which were the requirements of the project and how they were met.

4.1 Project overview

The TechDemo project was developed with some pre-defined ODD in mind, meaning that it is designed to work to work in specific environment and AD applications, in which the surroundings are known. Such applications include autonomous shuttle service, airport or facilities transportation system or Autonomous Valet Parking (AVP).

With that in mind, Autoware is the perfect software framework candidate, as it seeks to do exactly that.

The focus of this work will be on the **localization** component, so in the following its working principle will be briefly explained.

4.1.1 Autoware localization component

Autoware high modularity allows for disparate sensor architectures to be used within its framework. The sensor configuration we seek to integrate is mainly a **3D-LiDAR + point cloud map + vector map** configuration, which is particularly suitable for our application. Other sensors, such as GPS and IMU, are also integrated.

Figure 4.1 shows a diagram of the architecture.

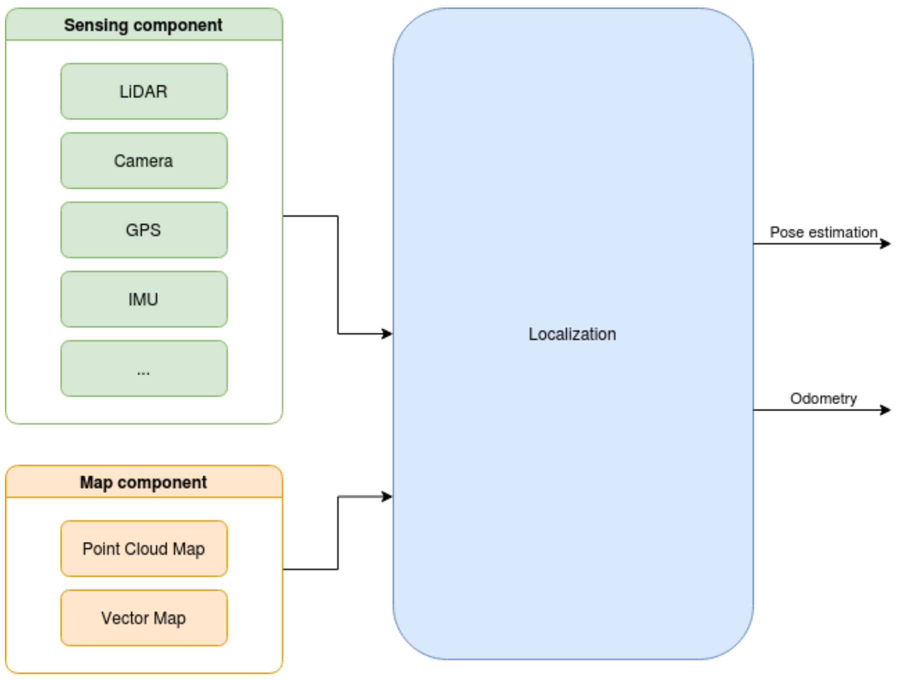


Figure 4.1: Localization architecture

With this architecture, Autoware relies on a pre-generated point cloud map which will be used as "ground truth" by scan-matching algorithms for pose estimation and odometry, as well as detecting changes and facilitate object detection and classification.

On a large scale, dedicated Mobile Mapping Systems (MMSs) are used to generate highly accurate point cloud maps. However, MMSs are notably expensive, since they require high-end sensors and systems. The aim of this work is to provide accurate enough pre-generated point cloud map, using SLAM algorithms and exploiting already owned sensors, greatly reducing the economic effort.

In the view of a real world case, the workflow would be the following:

1. Perform a 3D mapping of the surroundings by manually driving the vehicle in the working area
2. Process the obtained map, adding a vector map that includes roads and virtual traffic rules (traffic lights, road signs, etc.)
3. Autonomous operation

4.2 Geo-referenced map

A key requirement of the application was the need for a geo-referenced map, i.e. a point cloud map which points are expressed in global coordinates. This will allow for automatic localization when the vehicle is started and it can be particularly useful when the vehicle is designed to work in different ODDs to automatically know in which of them it is and automatically localize.

Although online generation of global-coordinates map is technically possible, using SLAM algorithms that use a GPS sensor, storing and dealing with big coordinates causes performance and accuracy issues.

At first, an attempt was made to develop software that would integrate this feature and generate an online geo-referenced map, but after the issues described above, it was agreed that it was more convenient to generate a local map and geo-reference it later with offline methods. To do so, different open-source software tools were used, including QGIS [15], CloudCompare [16] and the C++ library PDAL [17]. In some cases, geospatial data from Google Maps [4] was used.

The method used to obtain a geo-referenced map is the Ground Control Points (GCP) method.

4.2.1 GPC Method

The GCP method is a technique to geo-reference point cloud map by establishing reference points on the ground which GPS coordinates are known and matching them to corresponding points in the point cloud. Here is reported the workflow:

1. **Select (or put) GCP**

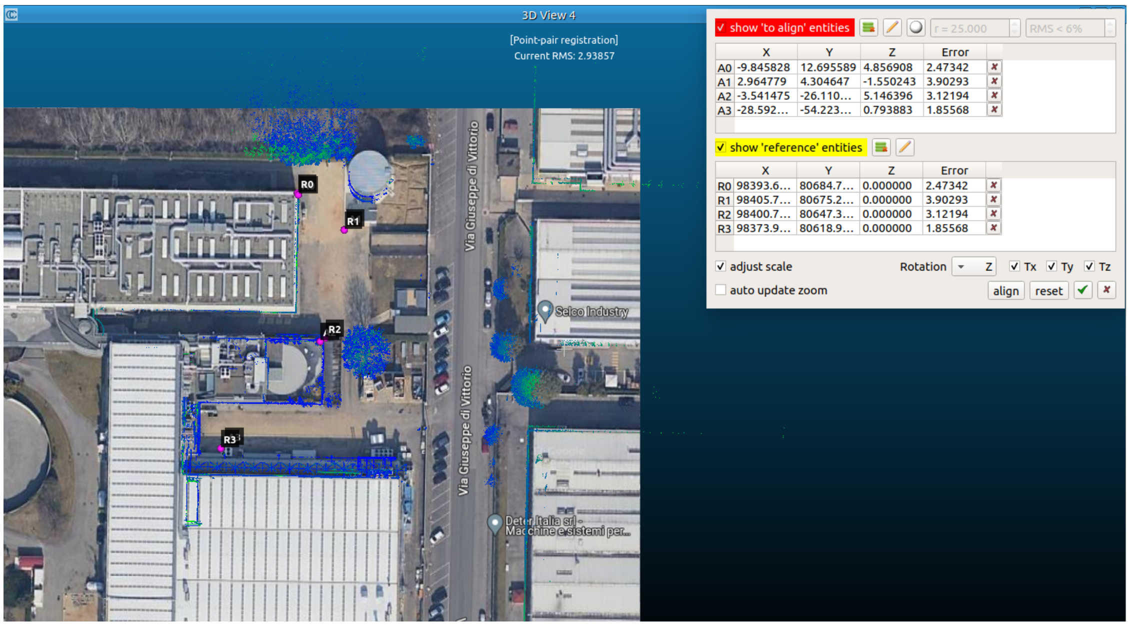
   A minimum of three reference points are selected in the area of the map. Those should be easily recognizable in the point cloud. If easily identifiable points are not available, they should be put for the purpouse (e.g. placing spheres).

2. **Measuring GCP**

   The GPS coordinates of the reference points should be measured. This was done by a mix of geospatial data from Google Maps and direct measurements with smartphone's GPS antenna.

3. **Matching GPS in the point cloud and transform it based on the coordinates**

The set of software tools reported in section 4.2 is used to perform this operation.



✓ show 'to align' entities

	X	Y	Z	Error
A0	-9.845828	12.695589	4.856908	2.47342
A1	2.964779	4.304647	-1.550243	3.90293
A2	-3.541475	-26.110...	5.146396	3.12194
A3	-28.592...	-54.223...	0.793883	1.85568

✓ show 'reference' entities

	X	Y	Z	Error
R0	98393.6...	80684.7...	0.000000	2.47342
R1	98405.7...	80675.2...	0.000000	3.90293
R2	98400.7...	80647.3...	0.000000	3.12194
R3	98373.9...	80618.9...	0.000000	1.85568

✓ adjust scale   Rotation Z   ✓ Tx ✓ Ty ✓ Tz

☐ auto update zoom

Figure 4.2: GCP in CloudCompare

Figure 4.2 shows the alignment of GCP between the point cloud and Google Maps performed with CloudCompare.

Chapter 5

Benchmark

This chapter delineates the process for testing and benchmarking the algorithms described in chapter 3. The discussion unfolds by explaining the general setting and workflow, followed by a qualitative analysis of each algorithm's performance. Finally, a comparative evaluation of the results is presented in section 5.6.

5.1 Setting and workflow

The setting and workflow structure is outlined as follows:

1. **Test scenario.** The first step was to define the test scenario. For the reasons described in chapter 4, and for convenience and repeatability, the open-air area surrounding the company, depicted in red in Figure 5.1, was selected as the working scenario. The original idea was to test in different areas, with different conditions and different paths. However, shortly after the beginning of this work, the access to company outdoor area was forbidden due to renovation. Fortunately, by that time, usable data was collected with the method described in item 2. For the sake of this work and our application, this scenario is sufficient, but for real-time and broader real-world applications, it is necessary to define more scenarios that should cover different conditions, such as lighting, road condition, dynamic object, etc.

2. **Data collection.** After the scenario was set, the subsequent step is to gather data from the sensors. This includes, for our purposes, scans from the lidar sensor, IMU measurements and GPS fixes. A ROS built-in tool called *Rosbag* is used, which allows to register messages published on each topic in a single

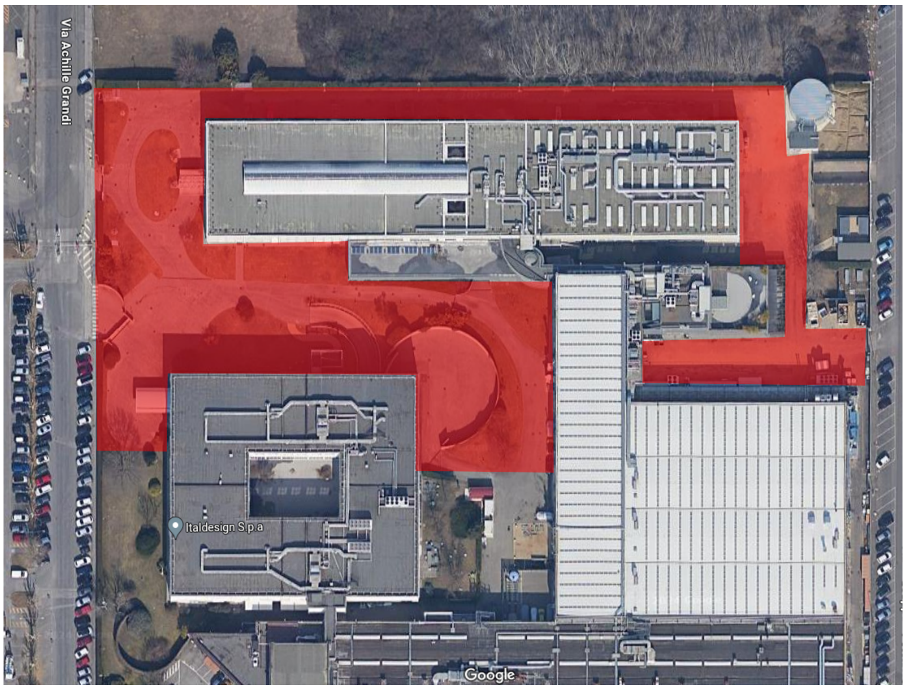


Figure 5.1: Aerial view of the working area [4]

file, called *Bag*. The *Bag* file can then be played, simulating the robot as if it was running in real time.

3. **Pre-processing.** In the next step, data is pre-processed to enable seamless and consistent testing for each algorithm. This include topic filtering to extract only the useful data, reducing file dimensions and computational cost, and data correction.

4. **Algorithm implementation.** In this step SLAM algorithms are implemented repeatedly to process the data and the output of each of them is saved.

5. **Results evaluation and comparison** The results coming from each algorithm are then analysed and compared in order to find the most suitable algorithm for the application.

The analysis of the results is focused on two factors:

- **Performance of the algorithm** in terms of run-time and computational cost, to evaluate feasibility on reduced power hardware.

The performance of the algorithms may depend from the power of the device in which they are ran. For portability and consistency, the algorithms were not tested on the AD processing unit of the TechDemo, but on a personal company laptop.

5.2 LIO-SAM

The first algorithm that was tested is LIO-SAM [2]. As stated in section 3.2, LIO-SAM allows to integrate GPS data in the factor graph, so the test was performed both with and without GPS data. For each case, several iterations of the test was carried out. For sake of simplicity and readability, only some relevant ones are reported here. However, the algorithm never failed and always delivered usable output.

5.2.1 Accuracy evaluation

Figure 5.2 shows the point cloud map obtained **without the use of GPS data**, compared with the aerial view of the area.

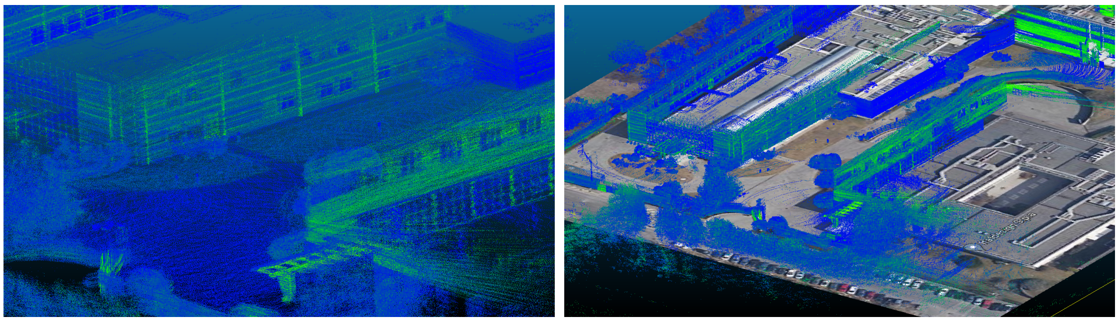


Figure 5.2: LIO-SAM: Generated map (no GPS)

Overall, the results were satisfactory. However, as shown in Figure 5.3, a strong misalignment in the left part of the map is noticeable, when compared to the aerial view. The experiment shown here is specifically chosen to highlight this aspect: in some of the other iterations of the same test, the misalignment is not present. However, in most of them it is, and in the view of automating the map generating procedure, this could cause concern and is therefore worth of notice.

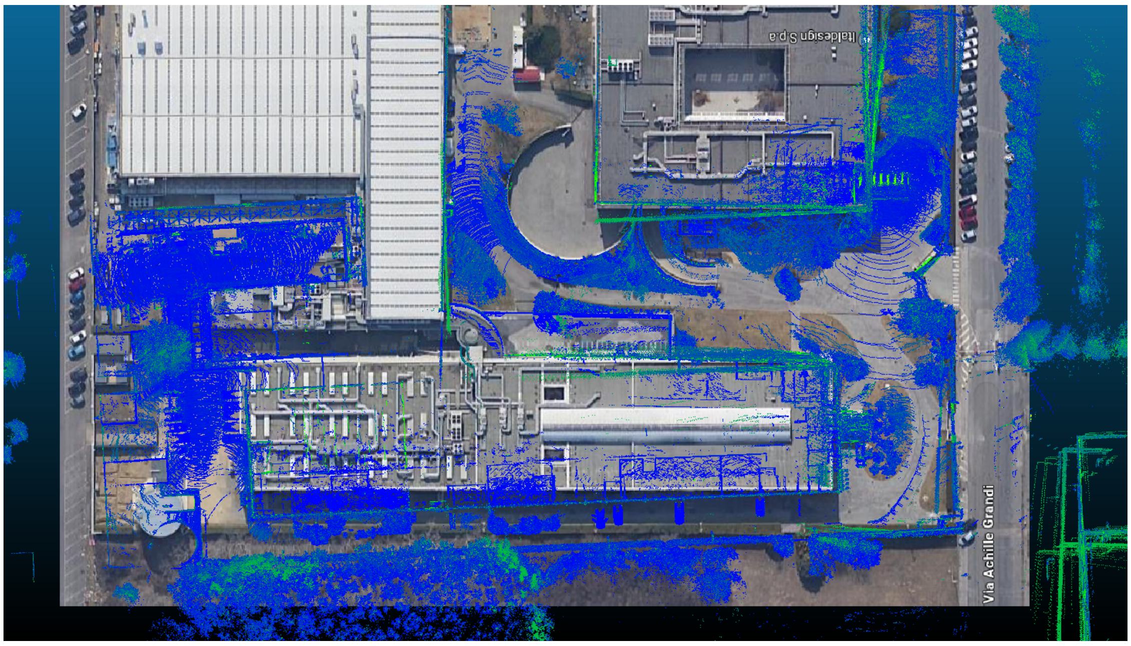


Figure 5.3: LIO-SAM: Generated map (no GPS) - top view

Enabling the **GPS factor**, the accuracy of the map greatly increase. An overview of the generated map is showed in Figure 5.4.

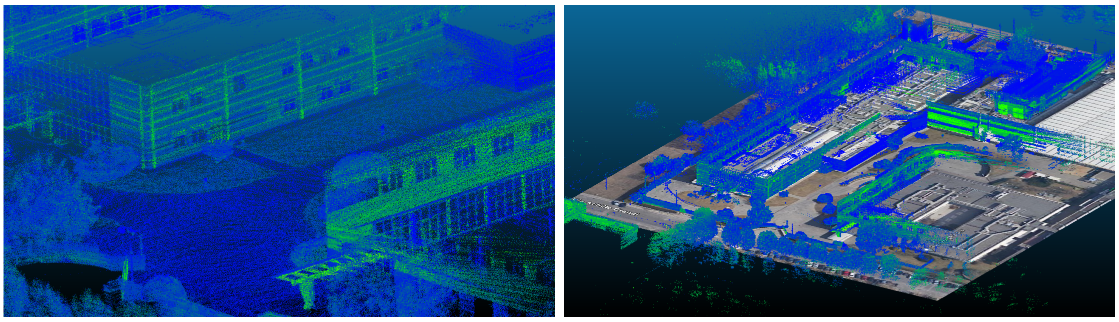


Figure 5.4: LIO-SAM: Generated map (GPS)

Figure 5.5 shows the top view of the map compared to the aerial view of the area. It is evident how, with GPS factor, the misalignment between the two almost disappear. In Figure 5.6, it is showed the comparison of the detail between GPS and no GPS.

The drift in the map is further showed in the trajectory plot in Figure 5.7.

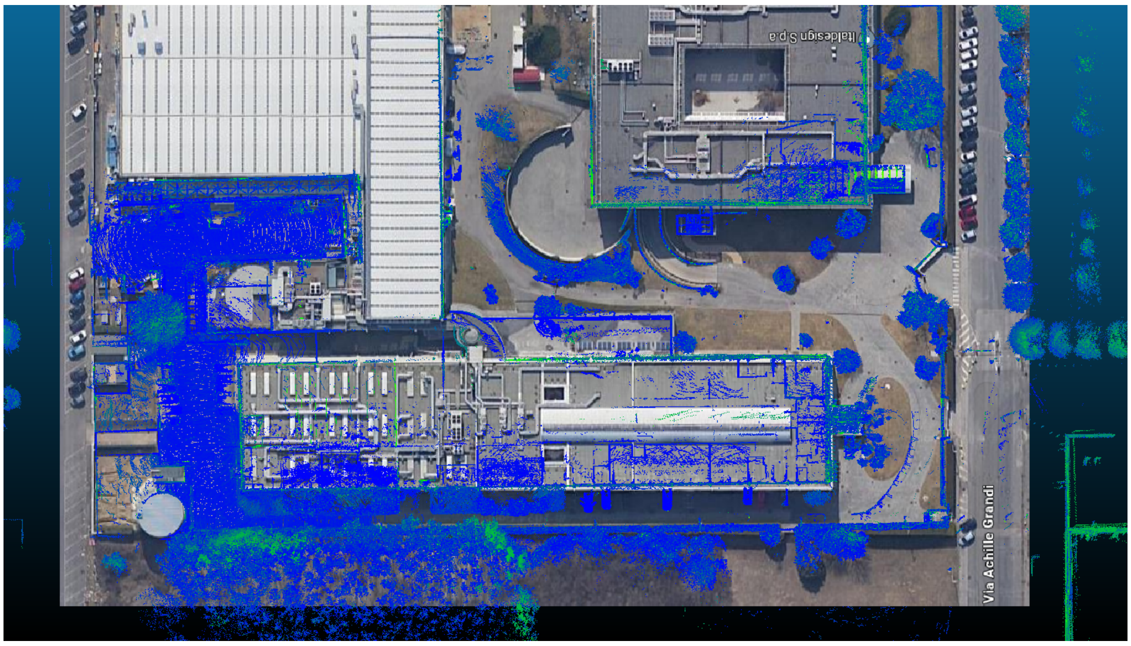
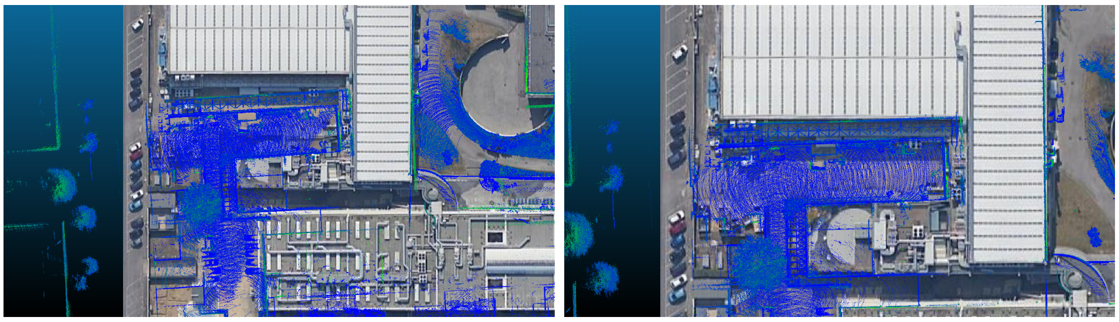


Figure 5.5: LIO-SAM: Generated map (GPS) - top view



(a) No GPS (b) GPS

Figure 5.6: LIO-SAM: No GPS vs. GPS

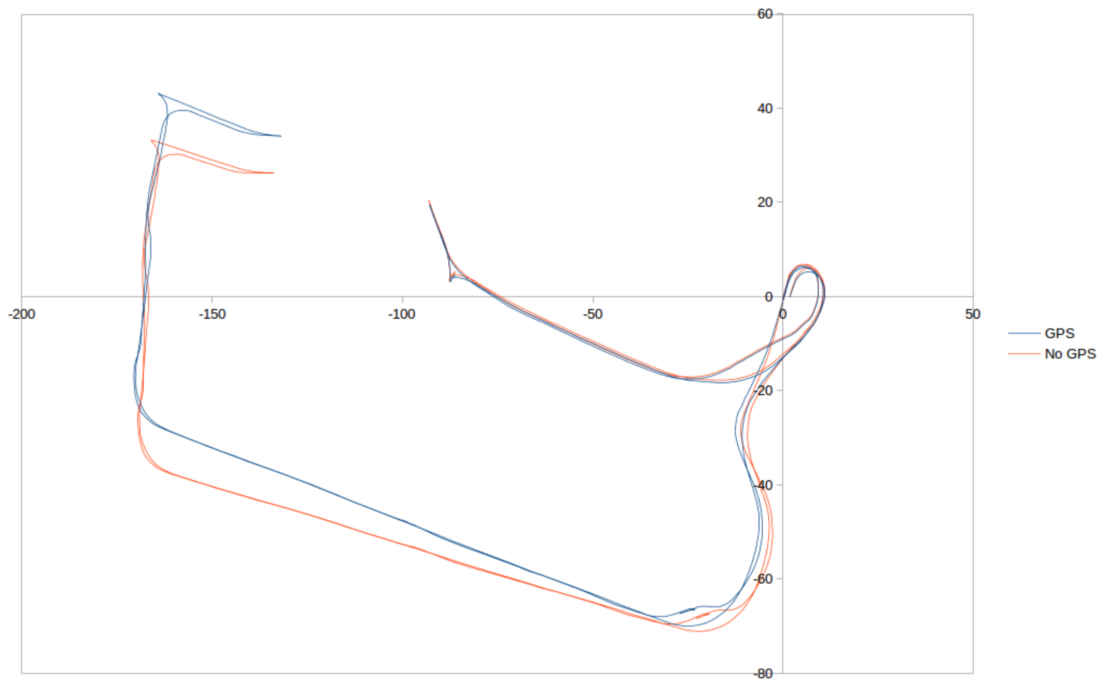


Figure 5.7: LIO-SAM: Trajectory plot

5.2.2 Performance evaluation

The performance of the algorithm was evaluated with two concepts:

1. **Single-scan processing time.** The average time necessary for processing a single lidar scan. Note that, since the lidar is delivering scans at 10 Hz, a processing time higher than 100 ms would require dropping frames for real-time usage.

2. **Stress test.** Taking inspiration from the authors of [2], a stress test was performed to evaluate the computational heaviness of the algorithms. The test consists in feeding data to the algorithm progressively faster than real-time and evaluate at which rate it would stop delivering satisfactory results.

The results of the performance tests are reported in Table 5.1

In both cases, the runtime to register a single lidar frame exceeds 100 ms. However, as stated in section 3.2, LIO-SAM do not need to process every scan thanks to its keyframe selection feature. This allows the algorithm to work in real-time.

Options	Runtime (ms)	Stress test
No GPS	111.2	11x
GPS	112.6	2x

Table 5.1: LIO-SAM: Performance results

There is irrelevant difference between the runtimes of the two cases, however, with GPS enabled, the algorithm failed the stress test just above 2x real time. This is probably due to the heavy graph update rather than processing time of a scan.

Nevertheless, LIO-SAM with GPS can still be used in real-time, and is preferable since it provides more accurate maps.

5.3 LeGO-LOAM

The next algorithm in test is LeGO-LOAM. Different tests were carried out and here is showed the best among them. However, results were extremely consistent among the tests and overall similar.

5.3.1 Accuracy evaluation

Figure 5.8 and Figure 5.9 show an overview of the generated map. The algorithm is able to provide satisfactory results but, due to the missing of a global measurement in the graph, an incremental drift on long distances, as shown in the left of Figure 5.9. Figure 5.10 shows the drift in detail.

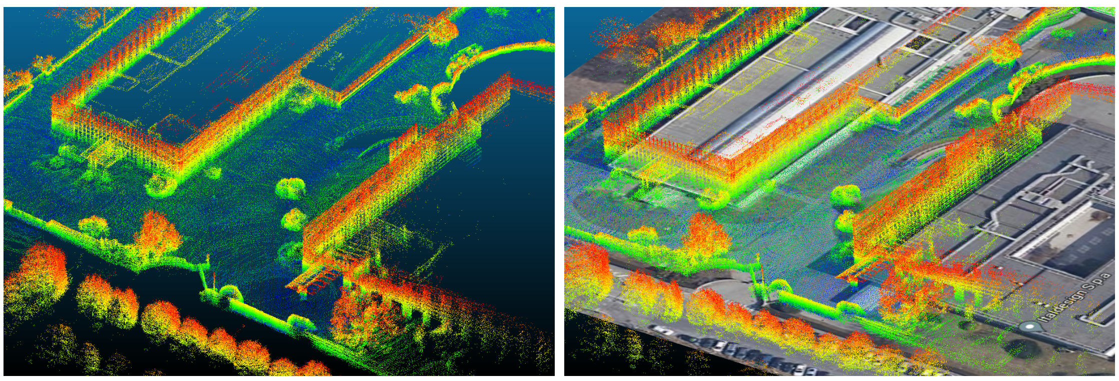


Figure 5.8: LeGO-LOAM: Generated map

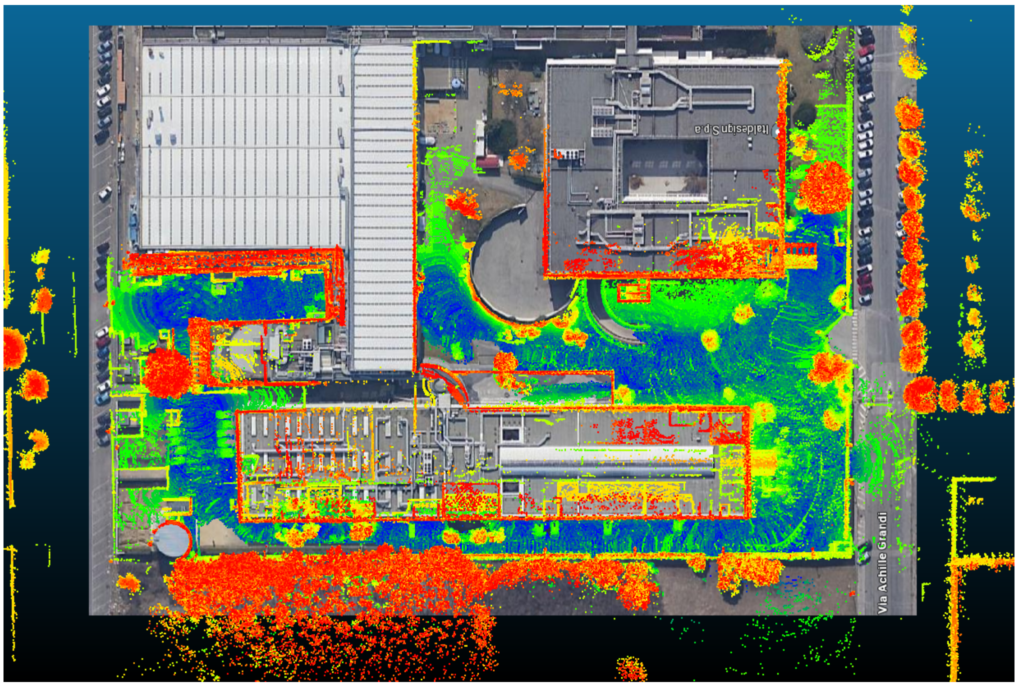


Figure 5.9: LeGO-LOAM: Top view

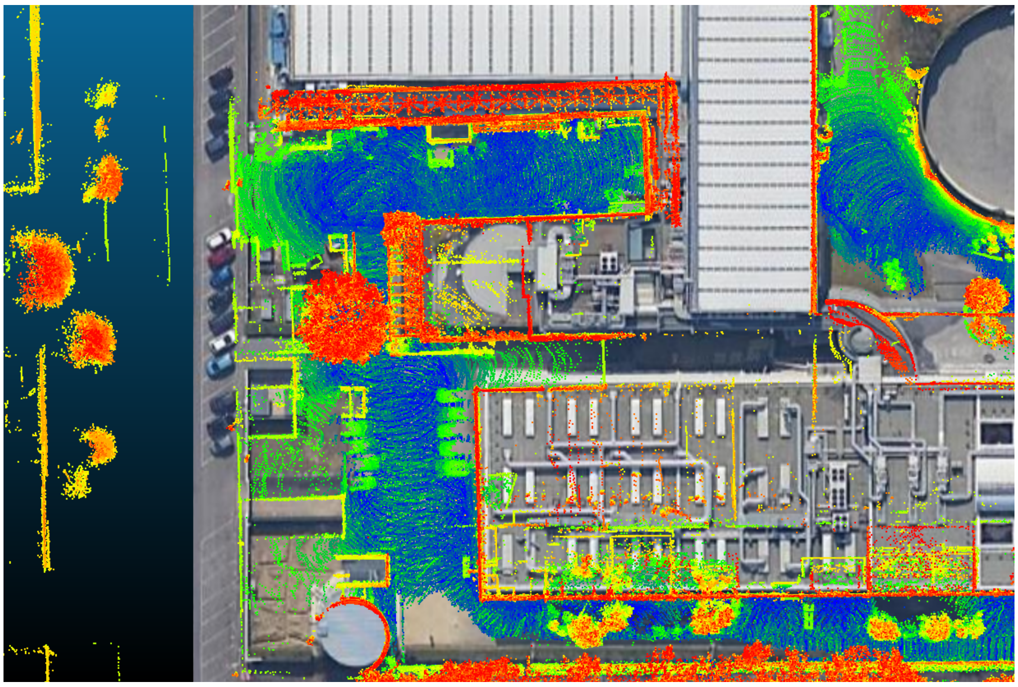


Figure 5.10: LeGO-LOAM: Drift detail

Moreover, the generated map shows a drift also in the z direction, resulting in a bent map in the same region. Figure 5.11 shows this behavior.

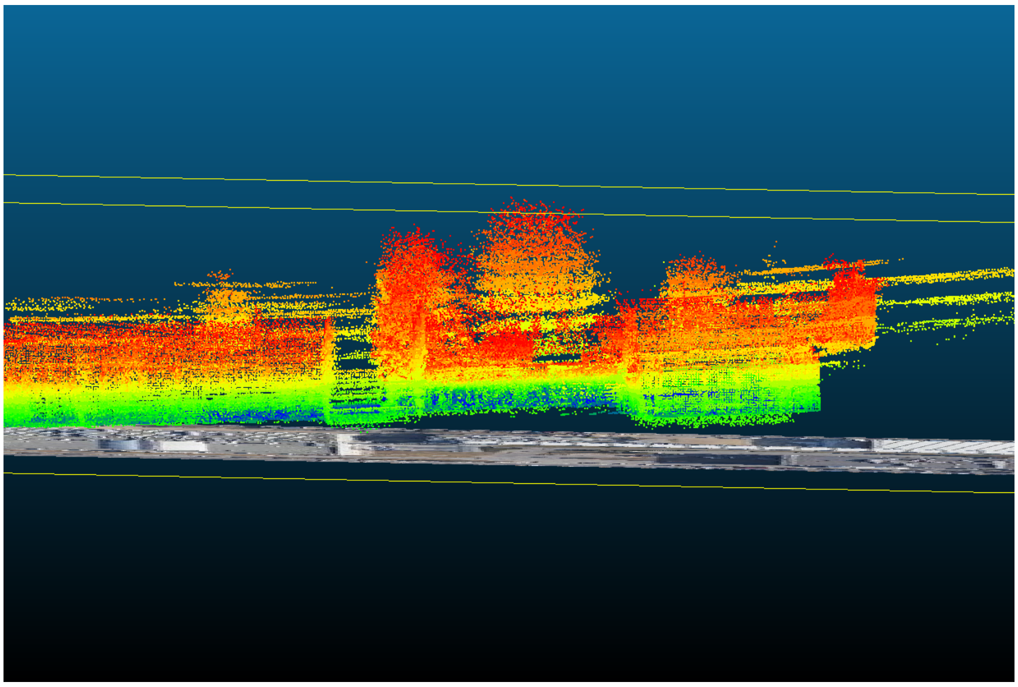


Figure 5.11: LeGO-LOAM: Bent detail

5.3.2 Performance evaluation

Similarly to 5.2.2, the performance is now evaluated by measuring the execution time of each lidar scan and feeding data faster than real time. Table 5.2 reports the obtained results.

Runtime (ms)	Stress test
157.1	17x

Table 5.2: LeGO-LOAM: Performance results

This algorithm performs really well in the stress test. However, the higher the feed rate is, the lower is the number of the registered points, reducing the density, and therefore resolution, of the map.

5.4 HDL Graph SLAM

As stated in section 3.4, this algorithm is highly modular and can be used with a wide set of sensor architectures. The tests were then performed in order to explore the different configurations. However, whenever an attempt to integrate data from the IMU was made, the algorithm would always fail. For that reason, none of the reported results include IMU data.

5.4.1 Accuracy evaluation

With only the lidar sensor active, the results are not satisfactory. As shown in Figure 5.12 and 5.13, the generated map is heavily bent and misaligned in more than one point, thus making it useless in any application.

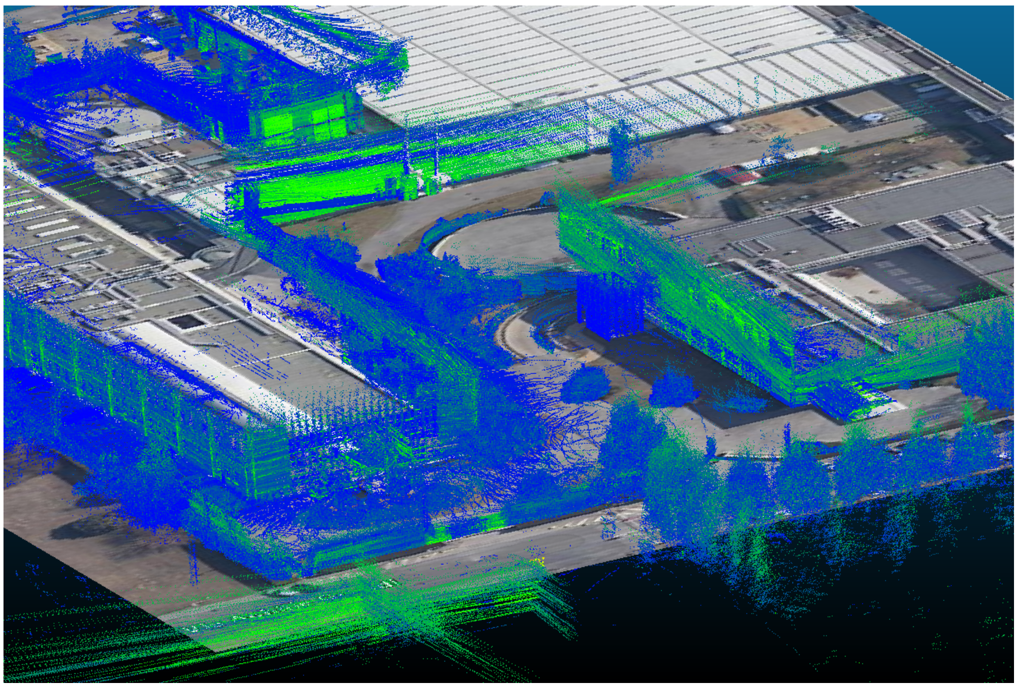


Figure 5.12: HDL Graph SLAM: Bent map detail

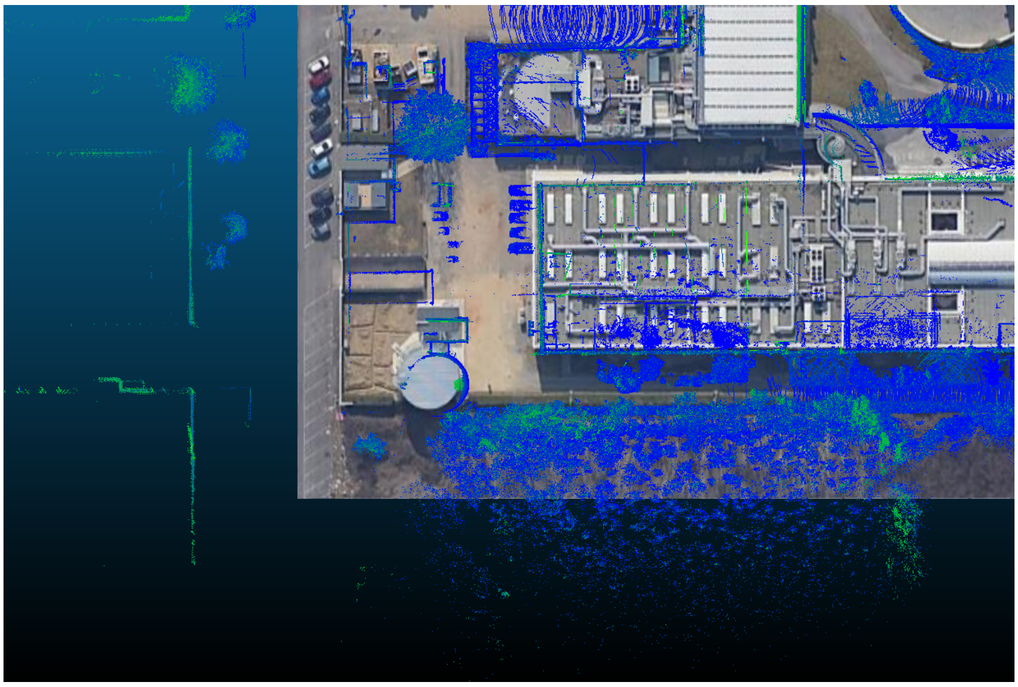


Figure 5.13: HDL Graph SLAM: Misalignment detail

The algorithm includes some solution to reduce the problem of bending, mainly **floor detection** and **GPS constraint**.

Firstly, floor detection is tested. The results concerning the flatness of the map are clearly better (Figure 5.14), but are still not perfect. On the other hand, the map is still greatly misaligned with respect to the aerial reference, as shown in Figure 5.15.

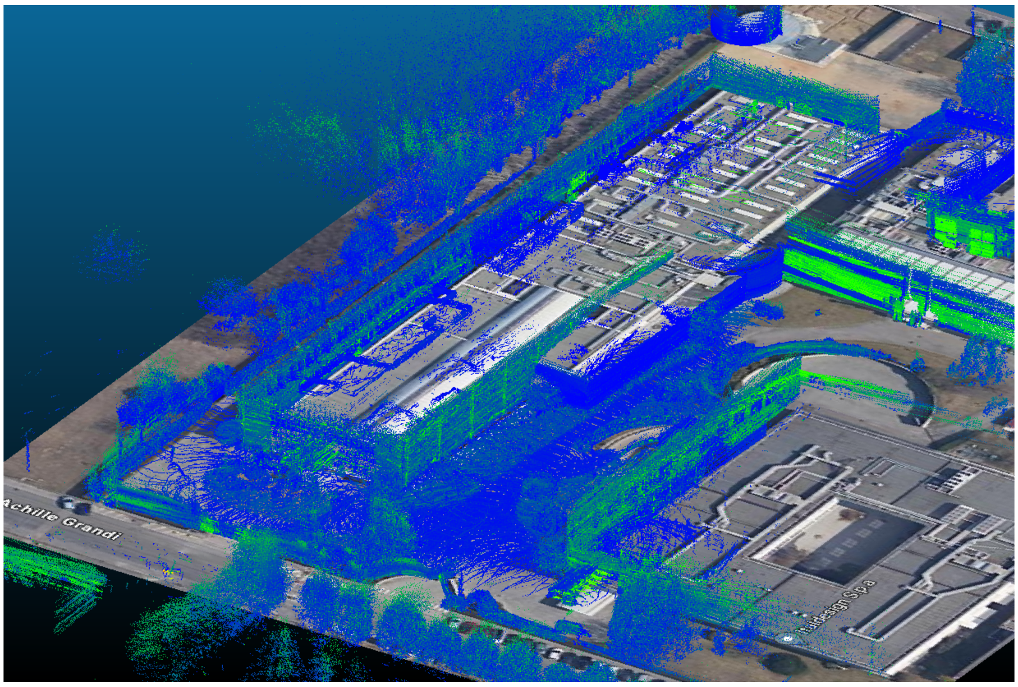


Figure 5.14: HDL Graph SLAM: Floor detection

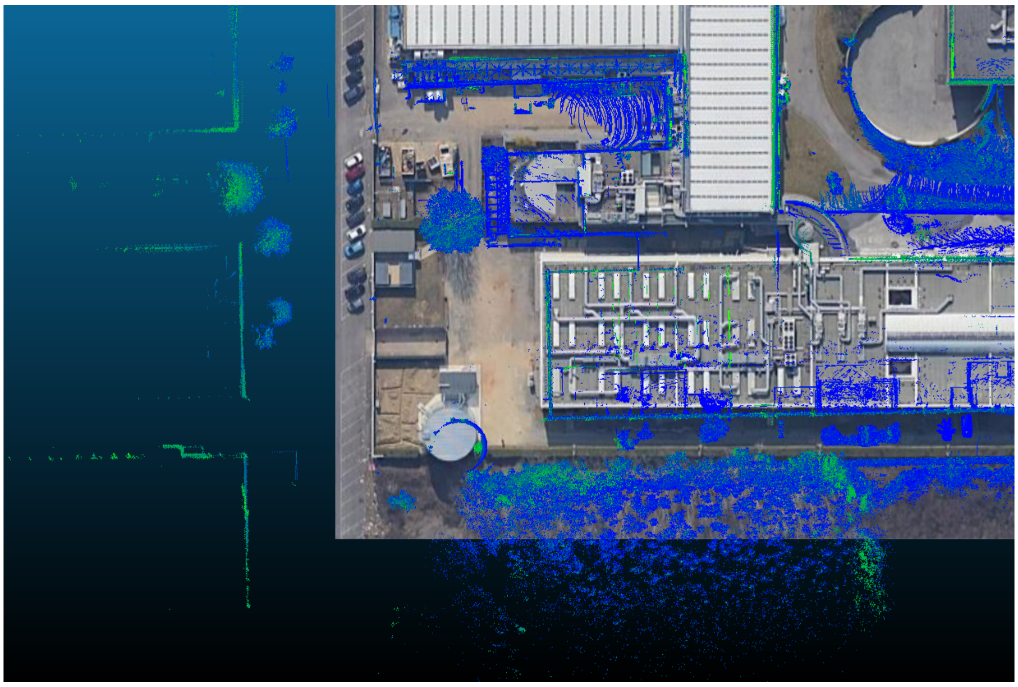


Figure 5.15: HDL Graph SLAM: Floor detection - Misalignment

With GPS constraint, the elevation issue disappears almost completely, with the exception of the area reported in Figure 5.16. However, the misalignment problem persists, as showed in Figure 5.17.

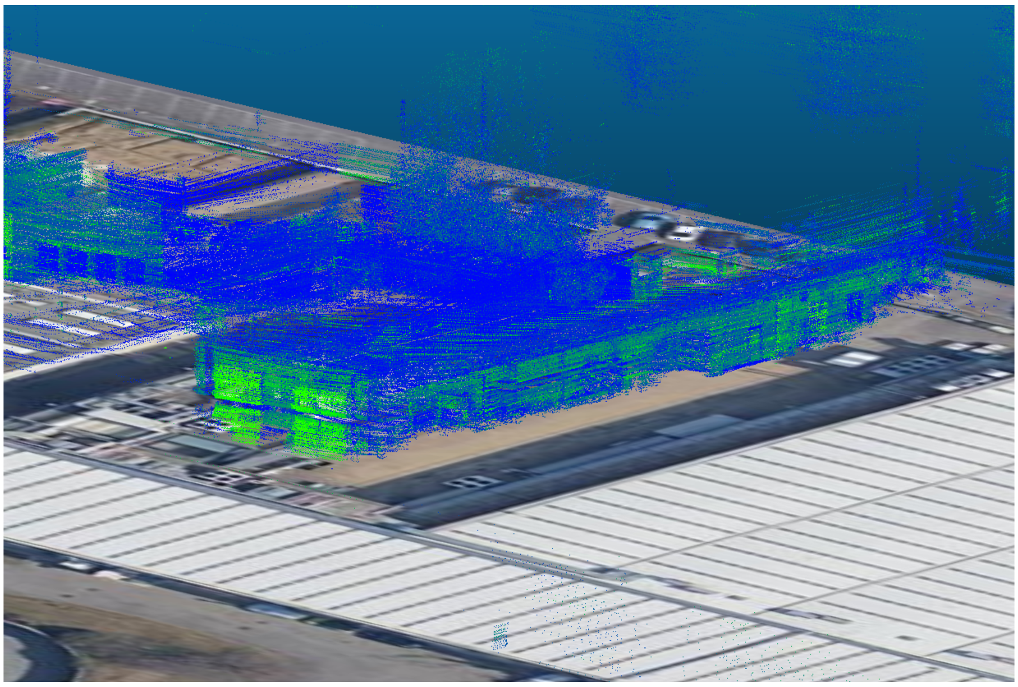


Figure 5.16: HDL Graph SLAM: GPS - Bending detail

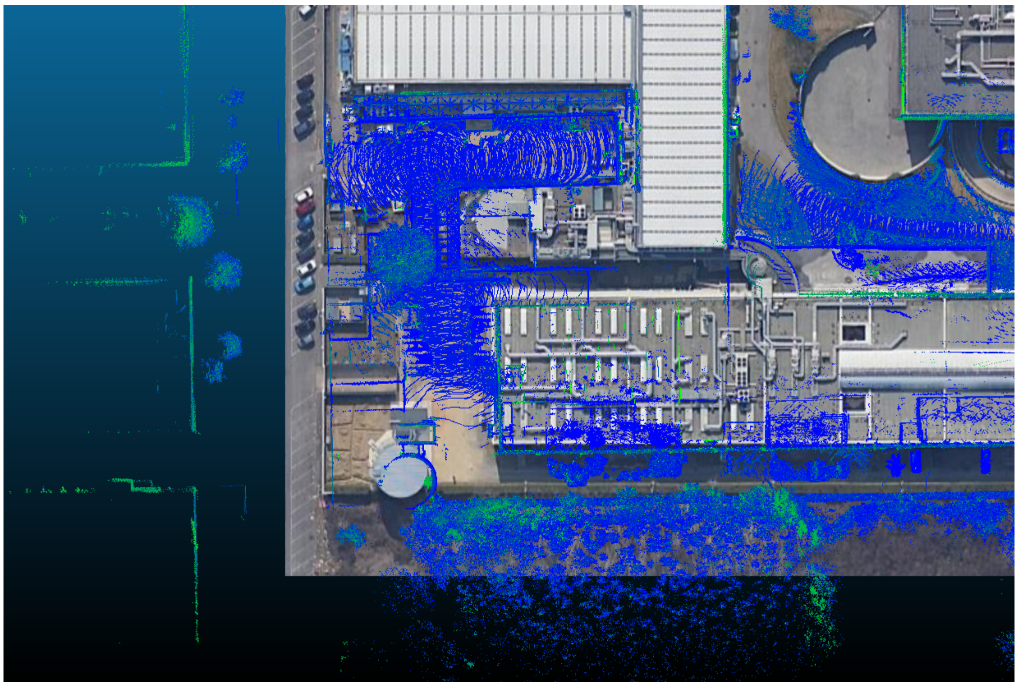


Figure 5.17: HDL Graph SLAM: GPS - Misalignment

The last test was performed enabling both GPS and floor detection constraints.

When used in conjunction, GPS and floor detection are able to provide great results in both directions. While the misalignment issue is completely solved (Figure 5.18 and 5.19), the flatness issue is still present (Figure 5.20). Nevertheless, it is a great improvement with respect to the first tests.

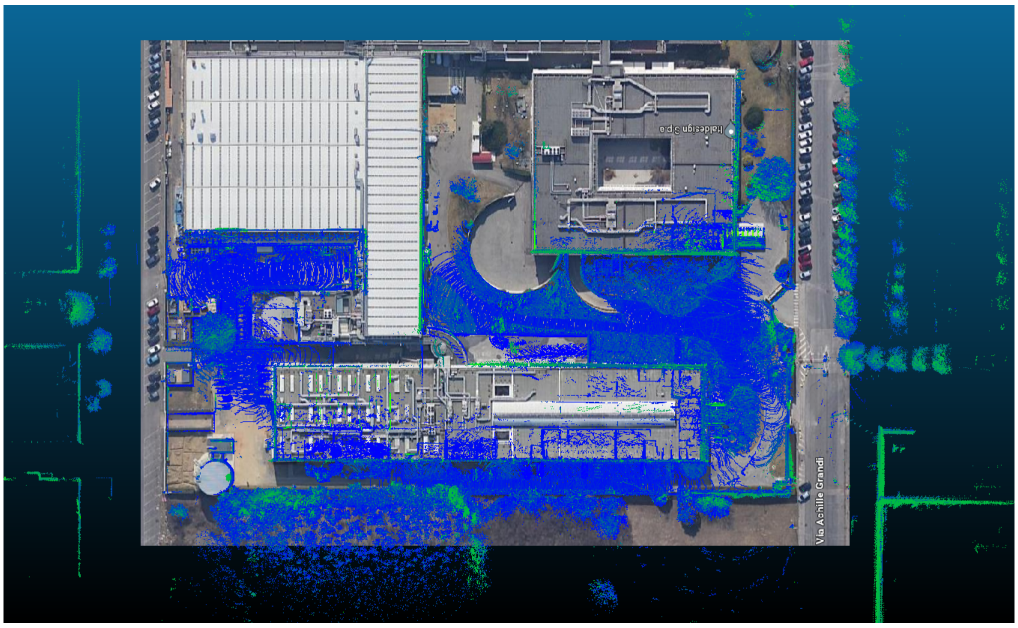
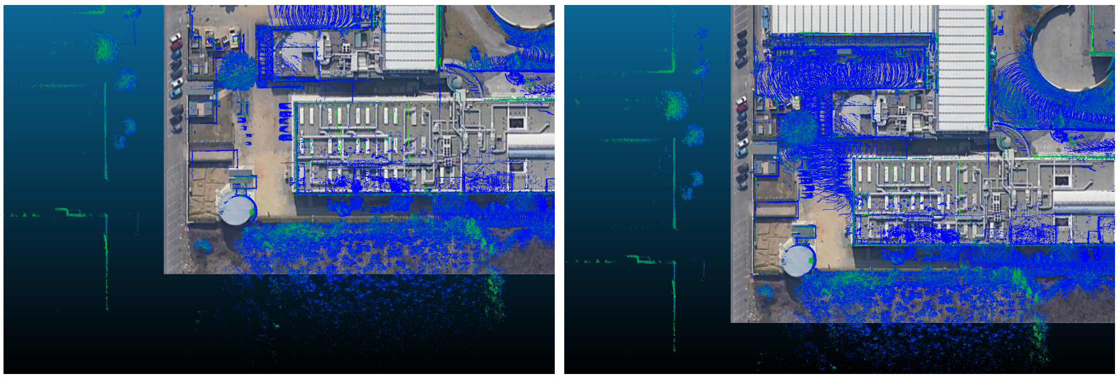


Figure 5.18: HDL Graph SLAM: GPS + Floor



(a) No GPS, no Floor detection

(b) GPS + Floor detection

Figure 5.19: HDL Graph SLAM: Alignment comparison

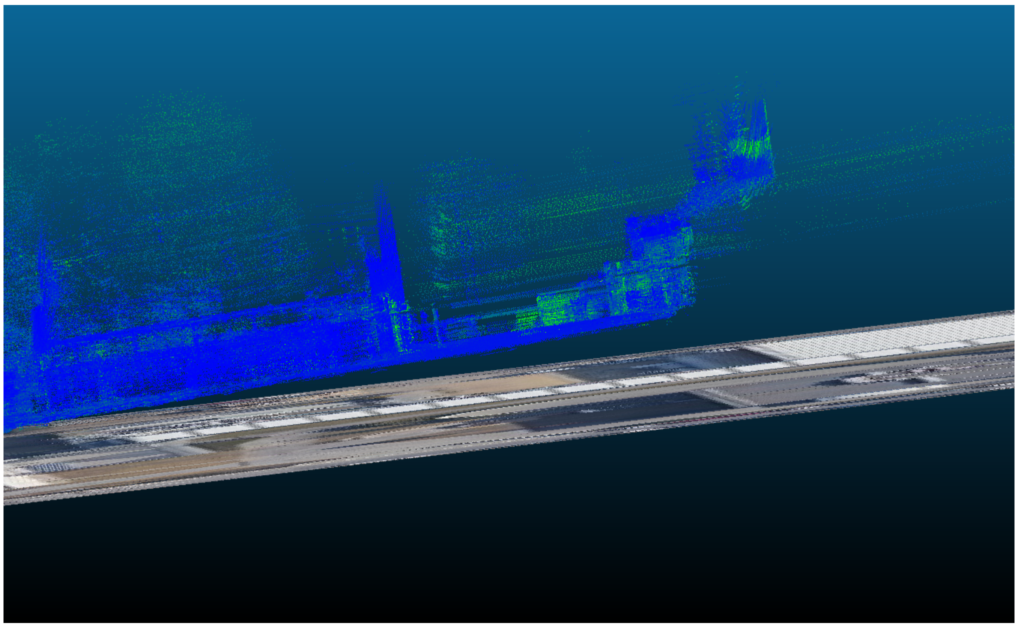


Figure 5.20: HDL Graph SLAM: GPS + Floor - Bending issue

5.4.2 Performance evaluation

Just like in subsection 5.2.2, the performance will be assessed by measuring the execution time of each lidar scan and feeding data at a pace faster than real time. Table 5.3 reports the results for each test.

Options	Runtime (ms)	Stress test
Only LiDAR	174.9	FAIL
LiDAR, Floor detection	180.4	FAIL
LiDAR, GPS	188.5	FAIL
LiDAR, GPS, Floor detection	197.5	FAIL

Table 5.3: HDL Graph SLAM: Performance results

The GPS constraint is particularly heavy, increasing greatly the runtime of each scan. In none of the cases the execution is below 100 ms, meaning that the algorithm is not able to process every lidar scan.

Interestingly, the algorithm failed the stress test in every condition. Nevertheless, as stated in section 3.4, the algorithm was never designed with performance in mind, so similar results should have been expected.

5.5 FAST-LIO

The last algorithm that was tested is FAST-LIO. The algorithm was designed to be lightweight and efficient and it provided outstanding results.

Figure 5.21 shows an overview of the generated map.

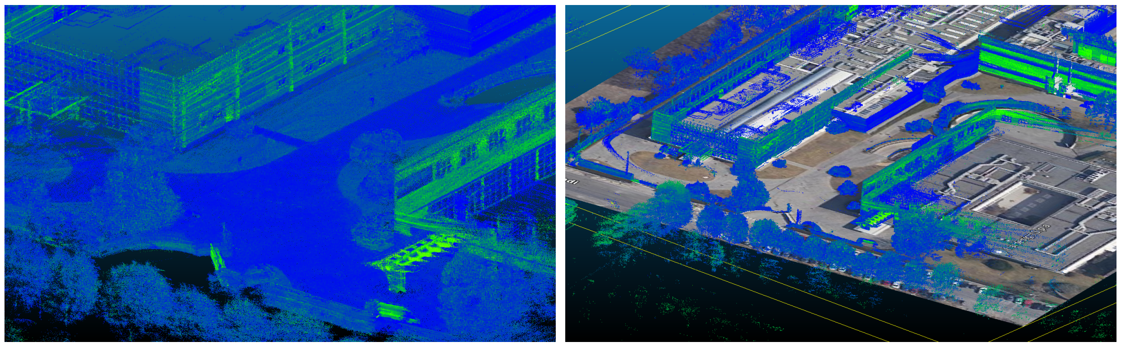


Figure 5.21: FAST-LIO: Generated map overview

5.5.1 Accuracy evaluation

The algorithm performed really well in terms of accuracy, showing reliable results in mapping accuracy when compared to the aerial view of the area (Figure 5.22).

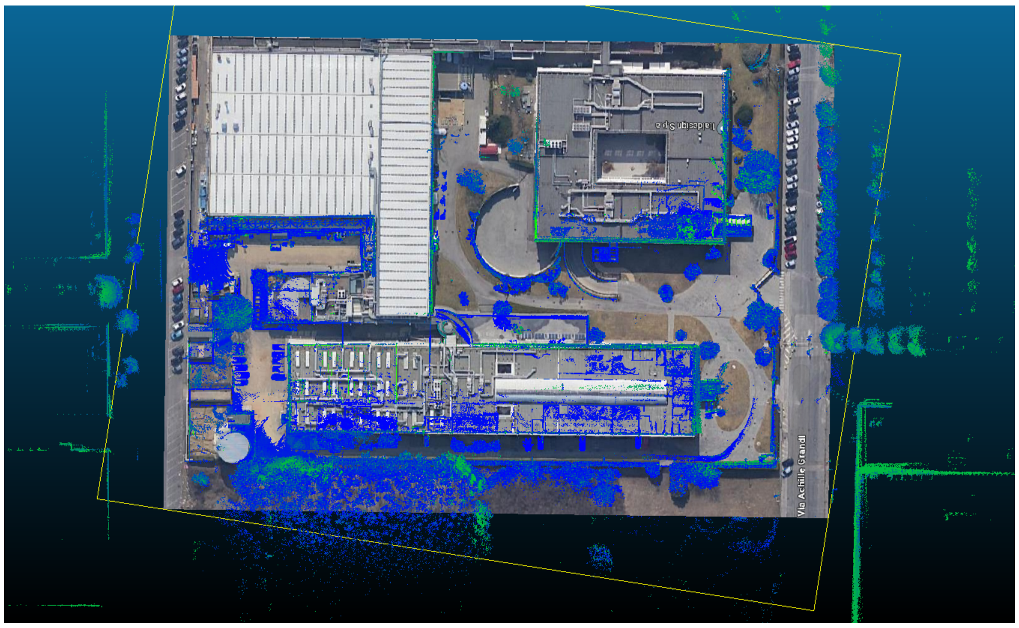


Figure 5.22: FAST-LIO: Top view

Unlike other algorithms, it didn't suffer from map bending. However, a slight misalignment of the map is present, as showed in Figure 5.23. Nevertheless, the algorithm provided incredible results, especially when considering its outstanding scan registration time reported in Table 5.4.

FAST-LIO also reported the highest point cloud density of all the tested algorithms.

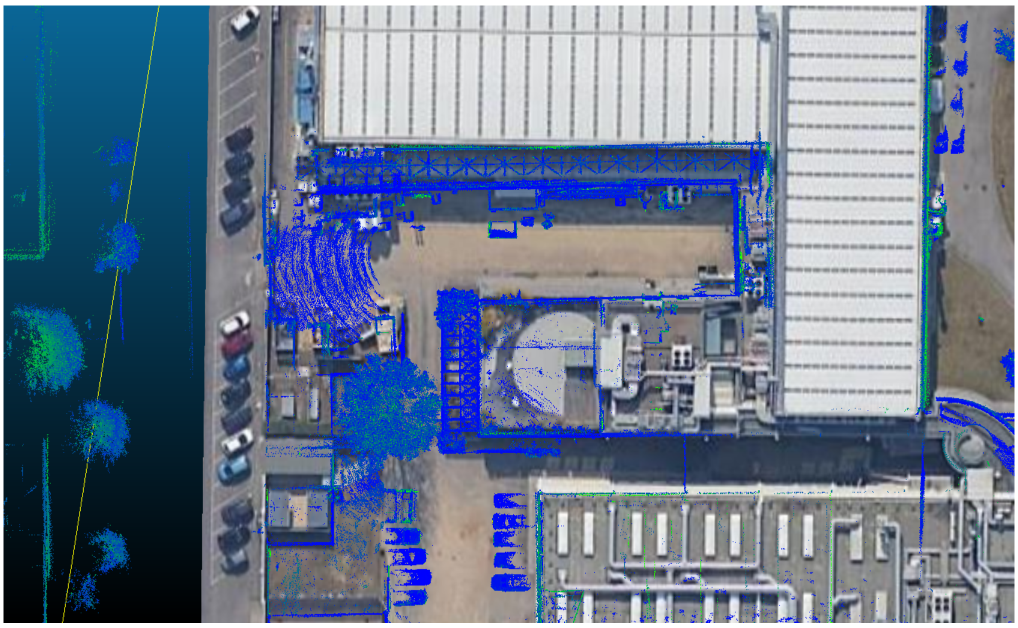


Figure 5.23: FAST-LIO: Misalignment detail

Although the generated map showed excellent results, the trajectory estimation was not as successful. Figure 5.24 shows the trajectory plot. It can be seen how the return path manifests a drift with respect to the first one, which was not present in the actual path.

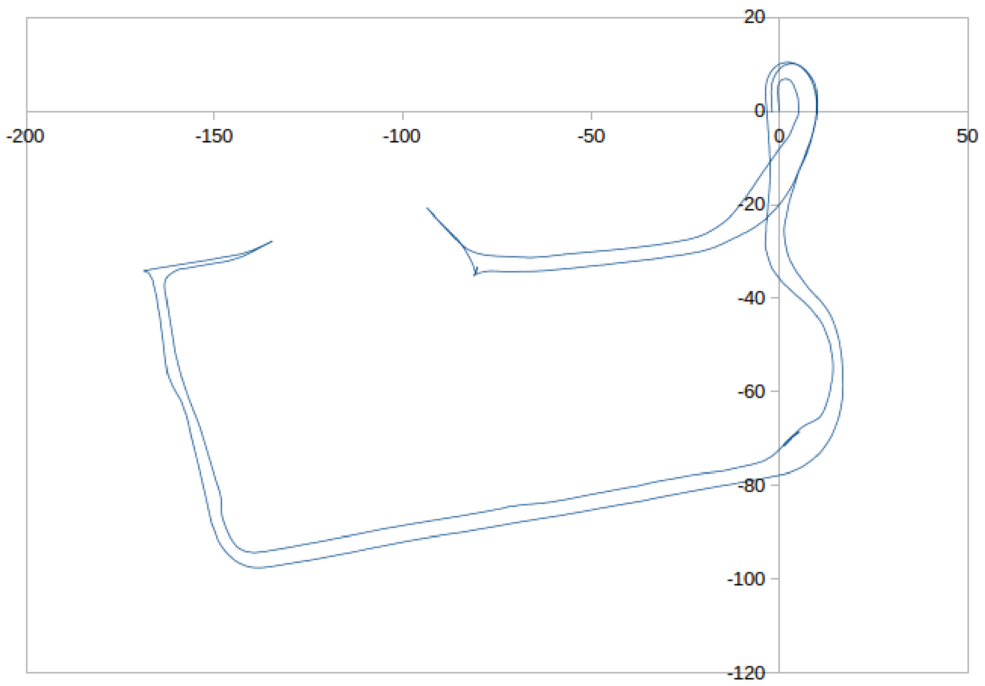


Figure 5.24: FAST-LIO: Trajectory plot

5.5.2 Performance evaluation

As stated before, FAST-LIO was designed to be computational efficient and lightweight. The results reported in Table 5.4 are a proof of that.

Runtime (ms)	Stress test
14.7	7x

Table 5.4: FAST-LIO: Performance results

The runtime for each scan is outstandingly low, an order of magnitude less than the other algorithms, allowing for true real-time operation with no dropped frames.

FAST-LIO cannot be forced to run in real time, meaning that if a new scan is available before the processing of the last one ends, it is simply put in queue, instead of being discarded, and it is processed whenever possible.

Therefore, the algorithm is able to process in real-time, according to the processing time reported in Table 5.4, valid for our set-up, for data rate up to 68 Hz, or about 7x in our stress test. However, as long as there is memory available in the processing device, the algorithm never fails even for feed rate higher than that.

5.6 Results

In this section, a comparison between the results of the proposed algorithms is presented.

1. **LIO-SAM**. LIO-SAM demonstrated robust performance in both mapping accuracy and computational efficiency, especially when using GPS. It is fast enough to allow the system to work in real-time, and it is able to generate high resolution maps.

2. **LeGO-LOAM**. LeGO-LOAM shows competitive mapping accuracy comparable to LIO-SAM. However, it showcased slightly higher computational heaviness, as well as greatly lesser dense generated maps. Nevertheless, the satisfactory results and its ability to work without an IMU make it flexible and a good candidate for a wide range of applications.

3. **HDL Graph SLAM**. The algorithm performs adequately when used in smaller environments, but it showed limitations both in mapping accuracy and computational needs. Indeed, the only usable map was possible to obtain only with large usage of the resources, almost double with respect to LIO-SAM.

4. **FAST-LIO**. FAST-LIO excelled in both processing speed and mapping accuracy, providing outstanding results in the generated map. It is the only algorithm, aside from LIO-SAM, which does not suffer from map bending, and shows lesser drift when compared to non-GPS aided algorithms. It also produced the denser point cloud among the tested algorithms.

Table 5.5 reports the results of each algorithm compared within each other. The values refer to the best test, in terms of mapping accuracy, between all the tests performed. A measurement of the density of the point cloud is also introduced, in the form of number of points of the generated map. The number reported below is the average of all of the tests between each algorithm.

Figure 5.25 shows a comparison between the trajectories obtained from each algorithm. Again, HDL Graph SLAM performed lesser than the other two. LIO-SAM and LeGO-LOAM show comparable results, leaning slightly towards LIO-SAM. However, this is remarkable from LeGO-LOAM, since it did not rely on GPS sensors. FAST-LIO, although provided one of the highest mapping accuracy, lacked in trajectory estimation.

Algorithm	Runtime (ms)	Stress test	Points
LIO-SAM	112.6	2x	7,353,209
LeGO-LOAM	157.1	17x	1,388,068
HDL Graph SLAM	197.5	FAIL	6,466,458
FAST-LIO	14.7	N/A	31,228,676

Table 5.5: Results comparison

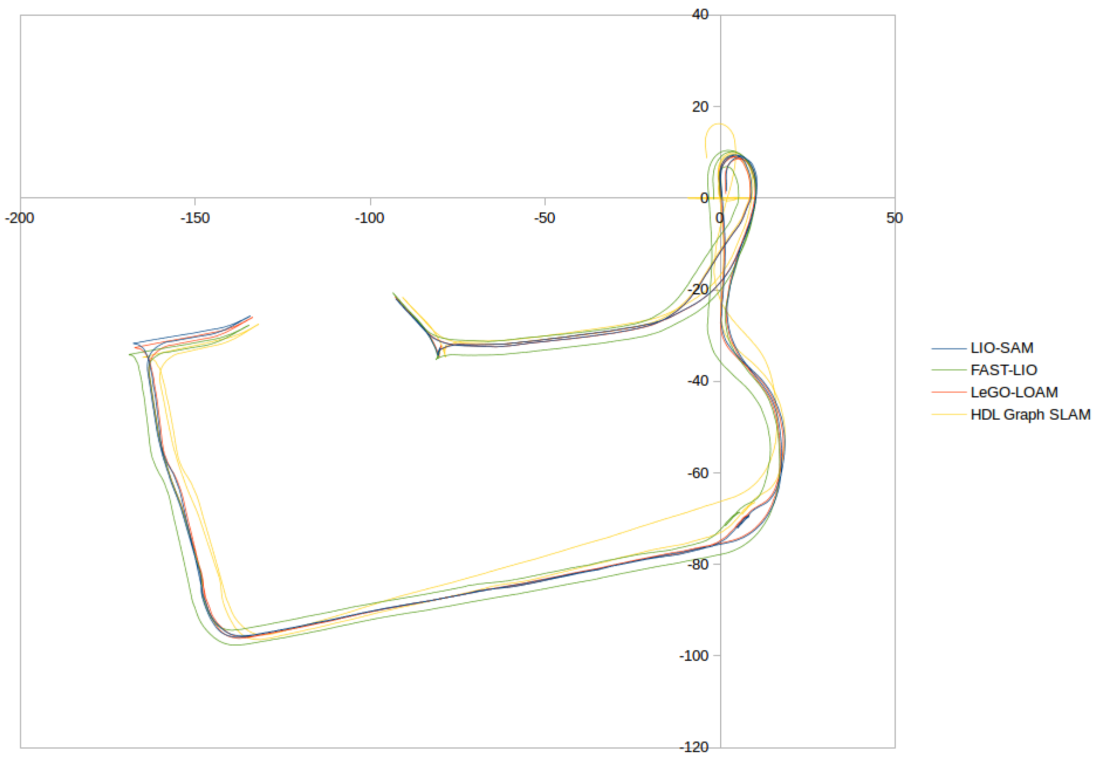


Figure 5.25: Trajectory comparison

LIO-SAM and FAST-LIO emerged as the top-performing algorithms, excelling in both processing time and mapping accuracy. While FAST-LIO demonstrated quicker processing speed and generated maps with higher density, it performed slightly worse in mapping accuracy due to the absence of a global measurement factor capable of rectifying trajectory drift. Given the critical importance of accuracy and reliability in the intended application, LIO-SAM emerged as the preferred choice. Additionally, the map resolution provided by LIO-SAM proved sufficient to yield satisfactory outcomes in the application. Consequently, LIO-SAM has been selected as the algorithm for integration into the TechDemo software.

Chapter 6

Conclusions and future work

In conclusion, this thesis explored various SLAM algorithms, offering an examination of their operational principles. Additionally, it investigates the realm of autonomous driving, explaining the functionalities of a tangible autonomous vehicle developed by Italdesign, known as the TechDemo prototype.

Each algorithm has been adapted to the hardware specifications and integrated into the TechDemo project. Extensive testing is conducted to assess the accuracy and computational efficiency of each algorithm, facilitating the identification of the optimal solution for the specified objectives.

While advancing through the project's development, a prospective idea emerged for future improvement in the project, briefly outlined in the subsequent section.

Mapping tool

Having in mind the workflow presented in 4.1.1, for the first and second step, the actual autonomous vehicle is not needed. In the view of a commercial application, perform these operation with the real vehicle could be expensive and not efficient.

For that reason a hand-held *mapping tool* concept has been thought off, of which a render is presented in Figure 6.1. The tool should ideally work with a reduced set of sensors, possibly only with a lidar. Further investigations on LeGO-LOAM (3.4) and HDL Graph SLAM (3.3) could satisfy this requirement. Moreover, the complete suite of HDL Graph SLAM could be used to recognize the person holding the tool and automatically remove it from the map. Lastly, FAST-LIO could be a suitable choice thanks to its computational efficiency and ability to work on less powerful embedded hardware.

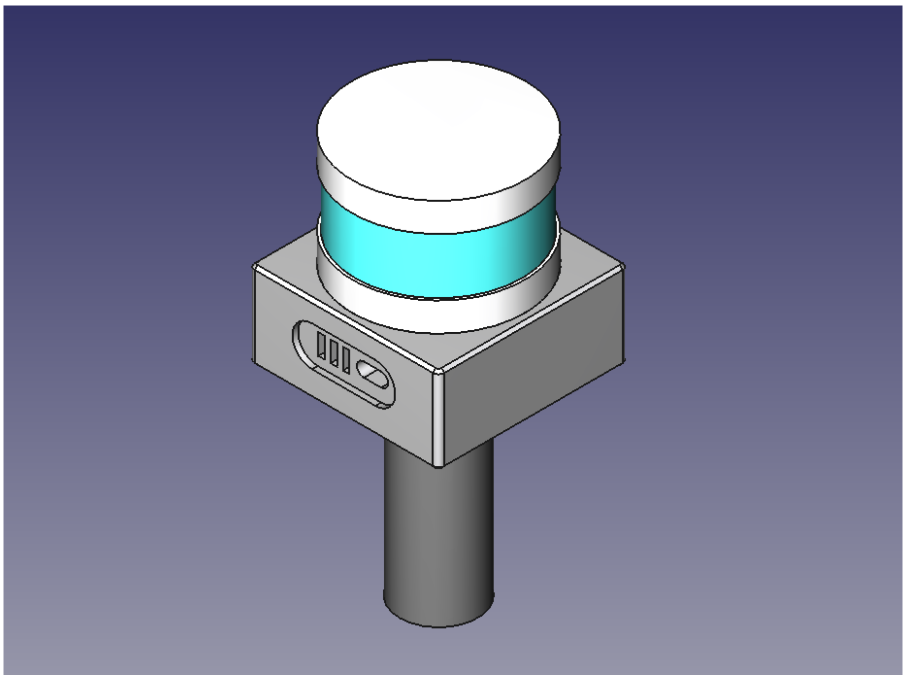


Figure 6.1: Mapping tool

Acronyms

AD Autonomous Driving. i, 1–3, 5–7, 11–14, 33, 39

ADAS Advanced Driving Assistance Systems. 1, 2

AI Artificial Intelligence. 1, 7, 8

AVP Autonomous Valet Parking. 33

BMS Battery Management System. 6

CAN Controller Area Network. 6, 12

CPU Central Processing Unit. 7

DMI Distance Measurement Unit. 10, 11

DOF Degrees of Freedom. 24, 26

ECU Electronic Control Unit. 6

EOL End-of-Life. 13

FAST-LIO Fast Lidar-Inertial Odometry. 17, 31, 55–57, 59, 60, 63

GCP Ground Control Points. v, 35, 36

GPS Global Positioning System. 9, 12, 19, 20, 22–24, 28, 30, 33, 35, 37, 40, 41, 44, 49, 51, 52, 54, 59

GPU Graphics Processing Unit. 7

IMU Inertia Measurement Unit. 10–12, 19–22, 24, 28, 31–33, 37, 48, 59

LeGO-LOAM	Lightweight and Ground Optimized Lidar Odometry and Mapping. 17, 24, 26, 28, 45, 59, 60, 63
LiDAR	Light Detection And Ranging. 8
LIO-SAM	Lidar-Inertial Odometry via Smoothing and Mapping. v, 17, 19–24, 28, 40, 43, 44, 59–61
MAP	Maximum a posteriori. 20
MMS	Mobile Mapping System. 34
NDT	Normal Distribution Transform. 29
ODD	Operational Design Domain. 3, 33, 35
OS	Operating System. 12
PCS	POS Computer System. 11, 12
POS	Position and Orientation System. 11
PSU	Power Supply Unit. 12
RADAR	Radio Detection And Ranging. 12
RANSAC	Random Sample Consensus. 30
ROS	Robot Operating System. 13, 14, 37
SAE	Society of Automotive Engineers. 2, 5
SLAM	Simultaneous Localization and Mapping. i, 3, 8, 11, 13, 17, 18, 24, 28, 29, 34, 35, 38, 59, 63
UDP	User Datagram Protocol. 6
UTM	Universal Transverse Mercator. 30

Bibliography

[1] On-Road Automated Driving (ORAD) Committee. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, apr 2021.

[2] Tixiao Shan, Brendan Englot, Drew Meyers, Wei Wang, Carlo Ratti, and Rus Daniela. Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020.

[3] Tixiao Shan and Brendan Englot. Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018.

[4] Google LLC. Google maps. `https://www.google.com/maps`, 2024.

[5] Leonardo Ferrari. Model control and low-level interface for an autonomous car prototype. October 2023.

[6] The Autoware Foundation. Autoware project. `https://autoware.org/`, 2024.

[7] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Transactions on Intelligent Transportation Systems Magazine*, 2, 12 2010.

[8] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. On-manifold preintegration for real-time visual–inertial odometry. *IEEE Transactions on Robotics*, 33(1), 2017.

[9] Ji Zhang and Sanjiv Singh. Low-drift and real-time lidar odometry and mapping. *Autonomous Robots*, 41, 02 2017.

[10] Igor Bogoslavskyi and Cyrill Stachniss. Fast range image-based segmentation of sparse 3d laser scans for online operation. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 163–169, 2016.

[11] Kenji Koide, Jun Miura, and Emanuele Menegatti. A portable three-dimensional lidar-based system for long-term and wide-area people behavior

measurement. *International Journal of Advanced Robotic Systems*, 16, 04 2019.

[12] Martin Magnusson, Achim Lilienthal, and Tom Duckett. Scan registration for autonomous mining vehicles using 3d-ndt. *Journal of Field Robotics*, 24(10), 2007.

[13] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6), jun 1981.

[14] Wei Xu and Fu Zhang. Fast-lio: A fast, robust lidar-inertial odometry package by tightly-coupled iterated kalman filter, 2021.

[15] The QGIS Project. Quantum geographic information system. https://qgis.org/en/site//, 2024.

[16] CloudCompare. Cloudcompare. https://www.danielgm.net/cc/, 2024.

[17] Andrew Bell et al. Point data abstraction library. hhttps://pdal.io/en/2.6.0/about.html, 2024.