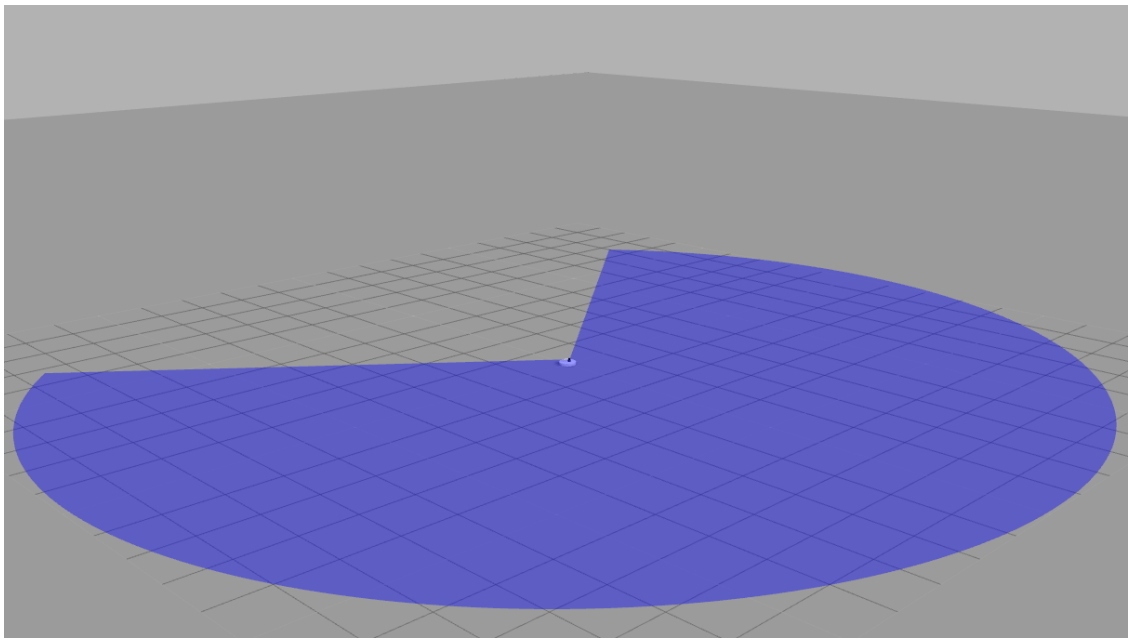
CHALMERS



Tracking an Object as a Vehicle Using Real-Time Data from a Fixed LIDAR Tower and Dual GPS Sensors

Optimizing and Tracking a Vehicle Localization

Bachelor's thesis in mechatronics engineering

Mustafa Almoghrabi
Daniel Bui

Mechatronics Engineering

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Bachelor's thesis 2023

Tracking an Object as a Vehicle Using Real-Time Data from a Fixed LIDAR Tower and Dual GPS Sensors

Optimizing and Tracking a Vehicle Localization

Mustafa Almoghrabi
Daniel Bui

CHALMERS

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2023

Tracking an Object as a Vehicle Using Real-Time Data from a Fixed LIDAR Tower and Dual GPS Sensors

Supervisor: Phd Henrik Fagrell , Diadrom
Examiner: Bertil Thomas,Professor at Automatic Control

Examensarbete 2023
Institutionen för Elektroteknik
Chalmers tekniska högskola
SE-412 96 Göteborg, Sverige 2023
Telephone +46 31 772 1000

Cover: Visualized data in Gazebo.

Abstract

Autonomous vehicles are used in various fields. They are typically positioned using GPS, but in many cases, GPS does not provide precise and accurate results,especially in proximity to obstacles such as walls or large metal containers in port environments.To overcome this, methods are needed to position a vehicle using real-time data from LIDAR and GPS. The purpose of this thesis is to analyze methods for positioning a vehicle using real-time data from LIDAR and GPS. There is relative freedom in choosing which method we will use.
In this study, a unique and new approach is implemented by using a Fixed LIDAR Tower in addition to a single GPS sensor installed in a vehicle. The collected data on the vehicle's precise location is transmitted to the Control Room at the Shipping Port for further analysis and monitoring.
Using the Extended Kalman Filter to combine data from multiple sensors, a new way to track and locate a vehicle is suggested and tested in a simulated environment. Different algorithms are compared and analyzed, mainly emphasizing the main emphasis being on effectiveness, computing efficiency, and applicability. The objective is to build a robust framework for a vehicle tracking system, improve autonomous navigation, and lay the groundwork for later testing and real-world application.

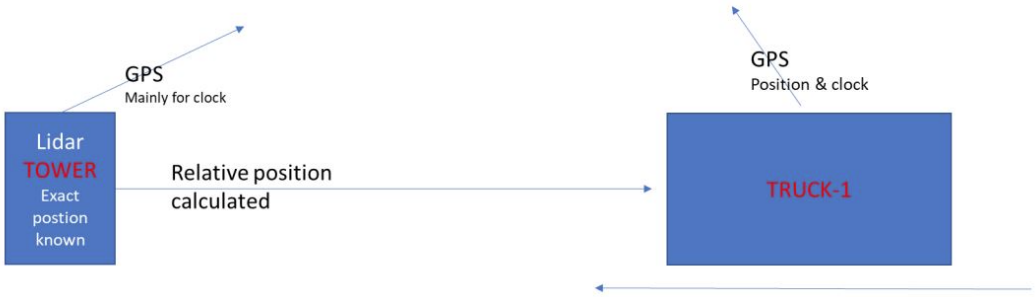


Figure 0.1: Sketch of the Preliminary System Design Concep

Keywords: Fixed LIDAR Tower, Dual GPS Sensors,Path planning,collision avoidance,Localization, Vehicle Location in Real-Time, Shipping Port Control Room.

Acknowledgments

We would like to express our sincere appreciation to Diadrom Company for providing us with the opportunity to work on this project. In particular,
Our deepest gratitude goes to our supervisor, Ph.D. Henrik Fagrell from Diadrom, for his invaluable guidance, wise advice, constructive criticism, and insightful comments. His expertise and mentorship have been instrumental in shaping the direction and outcomes of our project.
We would also like to extend our thanks to our coworkers at Diadrom, who offered us valuable assistance and advice during various stages of the project.
Furthermore, we would like to acknowledge our examiner Professor Bertil Thomas from Chalmers University of Technology. We greatly appreciate his valuable insights, evaluation, and recommendations, which have significantly contributed to the success of our project.
Finally, we are grateful to the university community and the existing literature in our field for their valuable contributions, which have facilitated the successful execution of our project.

Sammanfattning

Detta examensarbete fokuserar på optimering och spårning av ett fordon genom att använda realtidsdata från en fast LIDAR-torn och två GPS-sensorer. Fordon är beroende av GPS-positionering, men dess noggrannhet är begränsad i närvaro av hinder som väggar eller stora transportcontainrar i hamnen. För att hantera denna utmaning undersöks metoder som använder realtidsdata från LIDAR och GPS. Målet med raporten är att analysera olika metoder för att positionera ett fordon med hjälp av dessa sensorer. En ny och innovativ metod implementeras genom att kombinera en fast LIDAR-torn med GPS och en enkel GPS-sensor installerad i fordonet. Insamlade data om fordonets exakta position överförs för analys och användning av algoritmen. Extended Kalman Filter används för att kombinera data från flera sensorer och möjliggöra spårning och lokalisering av fordonet. Olika algoritmer jämförs och analyseras med fokus på effektivitet, beräkningshastighet och tillämplighet. Målet är att utveckla en ett fordonspårningssystem, förbättra autonom navigation och lägga grunden för framtida tester och verklig tillämpning. Systemets prestanda utvärderas med hjälp av Gazebo-simuleringsverktyget, vilket ger värdefulla insikter och möjliggör kontrollerade men dynamiska tester.Rapporten beaktar också möjligheter till systemoptimering, inklusive beräkningseffektivitet, minnesanvändning och svarstid. Genom att framgångsrikt simulera fordonspårning och lokaliseringsystemet demonstrerar rapporten hög noggrannhet och validerar effektiviteten hos den fasta LIDAR-tornet, de dubbla GPS-sensorerna och Extended Kalman Filter för att exakt spåra och lokalisera fordonet. Resultaten bidrar till området för autonom navigation och transporter och ger vägledning för ytterligare forskning och praktiska tillämpningar. Sammanfattningsvis lägger denna examensarbete en stark grund för utveckling och verklig implementering av fordonspårning och lokaliseringsystemet. Den simuleringbaserade metoden belyser systemets styrkor och begränsningar och banar väg för framtida optimering och förbättring. Projektets framgång bidrar till framsteg inom fordonspårning och lokaliseringsystem och skapar förutsättningar för framtida studier och implementeringar inom området för autonoma fordon.

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

DOP	Dilution of Precision
EKF	Extended Kalman Filter
GPS	Global Positioning System
LiDAR	Light Detectionand Ranging
MCL	Monte Carlo localization (Algorithm)
ROS	Robotics Operating System
SLAM	Simultaneous Localization and Mappin (Algorithm)
UTM	Universal Transverse Mercator
XML	Extensible Markup Language (text-based format)

Contents

List of Figures

1
Introduction

1.1 Background

Recent years have witnessed enormous progress in robotics and autonomous cars, particularly in localization and navigation technologies. The capacity to precisely and reliably determine the robot's or vehicle's location in its environment is a crucial concern in these disciplines. The Extended Kalman Filter (EKF) algorithm for sensor fusion is used in this thesis to create and implement a vehicle tracking and localization system in a simulated environment. Simultaneous Localization and Mapping (SLAM), Kalman Filter, Particle Filter (Monte Carlo Localization), Grid-based Localization, Feature-based Localization, and Visual Odometry are just a few of the localization techniques studied. Each has particular advantages and is suitable for various situations. However, due to its superior handling of non-linear models, adaptability, and effective real-time performance, we have chosen EKF. In this thesis, sensor fusion is used to overcome the limits of individual sensors and improve overall accuracy and resilience by combining measurements from many sensors, such as GPS and LIDAR. We can acquire helpful information and assess system performance in a controlled yet dynamic environment by deploying and testing our system within the Gazebo simulation tool, providing a solid foundation for real-world deployment. It is important to mention that the term "study" will be used instead of "project" in thesis.

1.2 Aim

This thesis employs the Extended Kalman Filter (EKF) for sensor fusion inside a simulated environment to develop and evaluate an effective vehicle tracking and localization system. The aim consists of the following:

- To support the selection of EKF based on the criteria of efficacy, computational efficiency, and suitability for the particular system, analyze and compare alternative localization techniques.
- To increase the precision and robustness of vehicle tracking and localization, design a system that effectively integrates several sensor inputs (namely GPS and LIDAR).
- Integrate this system into the Gazebo simulation tool to create a dynamic, realistic testing environment that delivers insightful data on system performance.

- Determine and address possible system enhancement and optimization opportunities, focusing on computing effectiveness, memory usage, and response time.
- Lay a solid foundation for the system's eventual deployment in the actual world and validation.
- Make an impact on the broader subject of autonomous navigation and transportation by contributing insightful research.
  These aims are intended to expand our knowledge of vehicle tracking and localization systems and open the door for further advancement and improvement in autonomous vehicle technology.

1.3 Delimitation

Delimitation are the limits or boundaries that the researchers set for themselves to keep their study focused and manageable. For this project, the following limits have been set:

- Simulation world: The project will be done in a simulated world with the help of ROS2 Foxy and Gazebo. Due to the time required to obtain a port entry permit, legal issues have been unavailable and have thus been considered outside the scope of this project. Even though the end goal is to use what was learned in the real world, this thesis will not include actual testing or use in the real world.
- Platform: Ubuntu 20.01 will be used as the platform for development and testing. This thesis doesn't cover how well it works or if it works with other operating systems.
- Vehicle: The project will focus on drivable cars as a way to use the system that is made and can be controlled.
- Sensors: The project will only use GPS and LIDAR to figure out where a car is. In this project, we won't look into other possible sensor systems, like RADAR or SONAR.
- Procedure for Calibration: The thesis will mostly be about getting two GPS devices (one on the self-driving car and one on the tower) to work together using Python and XML file types. Any other processes or methods for calibrating or synchronising are outside the scope of this work.
- Real-World Factors: The simulation will try to mimic real-world conditions as closely as possible. However, some real-world factors, like weather, signal interference, or physical obstacles, might not be fully reflected in the simulated environment.

2

Theory

2.1 LIDAR

A Lidar sensor is a device that utilizes laser light to measure distances and create 3D maps of the environment. Lidar stands for "Light Detection and Ranging." It operates by emitting laser pulses and measuring the time it takes for them to reflect back from objects, a process known as "time of flight" measurement. Lidar sensors find diverse applications in autonomous vehicles, robotics, environmental monitoring, surveying, and mapping, providing precise and detailed 3D information about surrounding objects.

The main components of a Lidar sensor include:

- **Laser Source:** A powerful laser that emits short pulses of light, typically in the infrared range. The laser serves to supply the necessary energy for accurate distance measurement.
- **Optics:** Lenses and mirrors that direct and focus the laser beam.
- **Scanning Mechanism:** A device that enables the Lidar sensor to scan the environment from different angles. This can be achieved through various techniques, such as rotating mirrors, oscillating mirrors, or beam steering technologies.
- **Photodetector:** A sensor that detects the reflected laser light, measuring its intensity to determine distance and capturing time of flight information.
- **Signal Processing:** These components control the timing of the laser pulses, measure the time it takes for the light to return, and process the collected data.

2.2 GPS

In this study, the Global Positioning System (GPS) played a vital role in tracking and analyzing location data. Precise geographical coordinates were obtained through GPS, which were then converted to Universal Transverse Mercator (UTM) for easier manipulation and interpretation [1]. The GPS data was received and processed, with a focus on ensuring the accuracy and reliability of the readings. High-quality GPS modules were employed, and appropriate filtering algorithms were implemented to achieve this goal. In the method section will provide a more detailed explanation about that.

2.3 Algorithm

The algorithm employed in this research study involves the fusion of GPS and LiDAR sensor data to accurately locate vehicles at the dock of cargo ships. Sensor fusion plays a critical role in enhancing the reliability and robustness of the acquired data for precise vehicle localization. The GPS data is processed and converted into UTM coordinates, while the LiDAR data aids in the identification of objects in the docking environment. By combining these two datasets, the algorithm enables the accurate localization of vehicles in the cargo ship dock environment. The development activities were performed on an Ubuntu 20.04 system, chosen for its compatibility with the ROS2 Foxy environment and its widespread use within the robotics community.

2.4 Programming languages and tools

2.4.1 Python and C++

The software components of the project were developed using Python, a high-level, interpreted programming language. Python's popularity for scientific computing tasks, including data analysis and machine learning, is attributed to its readability and extensive standard library. Additionally, the program was built using the C++ programming language, leveraging its efficiency and performance advantages. Furthermore, the integration with ROS2 was facilitated by the Python language library, enabling the harnessing of the robust capabilities of both languages for the project. To further aid our development, we utilized several specialized libraries and frameworks. These included:

- **ROS2 Foxy:** ROS (Robot Operating System) is an open-source, meta-operating system for robots [2]. We used the ROS2 Foxy version, which provides libraries and tools to help software developers create robot applications. It simplifies managing distributed computations that are often necessary in robotic systems.
- **rclpy** This is a Python library for ROS 2. It allows for creating Python nodes in a ROS 2 environment and provides the necessary tools to develop publishers, subscribers, services, and clients[3].
- **UTM:** This Python library converts latitude and longitude into UTM coordinates. UTM (Universal Transverse Mercator) coordinates are more practical for calculating distances and are widely used in GIS systems.

These programming languages and libraries played a crucial role in the development of our project, enabling efficient and effective implementation of various functionalities and ensuring compatibility with the required frameworks and systems.

2.4.2 Robot operating system

ROS (Robot Operating System) is an open-source framework designed for developing robot applications. It provides a modular architecture that facilitates communication and coordination between different robot components. The advanced version of ROS, called ROS2, offers seamless integration of sensors in robot applications.

Its modular architecture and improved performance enable efficient communication and coordination between sensor components and other parts of the robotic system. ROS2 offers a wide range of sensor-related packages and libraries for processing sensor data, calibration, and sensor fusion. This makes it a suitable choice for projects involving sensor integration.
By leveraging the capabilities of ROS2, autonomous vehicles can effectively utilize sensor data for navigation, obstacle detection, object tracking, and ensuring safe and efficient operation. The modular architecture, improved performance, and support for real-time communication in ROS2 make it an ideal platform for developing sensor-rich autonomous vehicle systems.

2.4.3 Gazebo

Gazebo is an open-source multi-robot simulation environment that serves as a platform for simulating and testing robot behavior in complex and realistic environments. It allows users to create virtual worlds, design and control robots, and evaluate their performance within simulated environments.
The software offers a comprehensive set of features and tools for simulating physics, dynamics, sensors, and actuators. This enables developers to test and refine robot designs and algorithms without the need for physical prototypes. Gazebo also supports the integration of various sensors and control systems, allowing for realistic simulations.
One of the notable aspects of Gazebo is its graphical interface, which provides a visual environment for users to construct virtual worlds, define robot models, and specify desired robot behaviors. Additionally, Gazebo offers a wide range of APIs and plugins that facilitate customization and extension, allowing users to incorporate their own algorithms and models into the simulation.
The simulations conducted in Gazebo have numerous applications, including the development and debugging of robot control software, testing navigation algorithms, evaluating sensor performance, and simulating complex real-world scenarios. It serves as a valuable tool for researchers, developers, and hobbyists in the field of robotics, providing a realistic and efficient platform for experimentation and analysis.

2.5 System Architecture

2.5.1 Scenario

The system architecture consists of a fixed Lidar sensor tower, dual GPS sensors, and the UTM algorithm. The lidar sensor tower provides high-resolution 3D point cloud data for object detection and mapping. The dual GPS sensors offer continuous positioning information. The UTM algorithm enables precise coordinate transformation between latitude-longitude coordinates and UTM coordinates. Additionally, the information and coordinates of the vehicle's location will be sent to the Shipping Port Control Room for monitoring and control purposes. This integration allows for effective tracking and management of the vehicle's position within the port area.

By incorporating the fixed lidar sensor tower, dual GPS sensors, UTM and Kalman Filter, the system ensures accurate and reliable vehicle localization. The combined data from these components, along with the transmission of information to the Shipping Port Control Room, enhances situational awareness and facilitates efficient operations within the port environment.

3

Methods

3.1 Synchronization

Synchronization refers to the coordination or alignment of multiple entities or processes to work together in a harmonized manner. It ensures that various elements or actions occur in a desired order or at a specific time, enabling them to cooperate effectively and achieve a common objective. The synchronization implemented in this project can be seen through two different scenarios:

a. General b. Case specific

3.1.1 General case

In the general case,the project scenario can be seen as a vehicle with a GPS sensor moving in a provided environment, the GPS will provide the X, Y, and Z coordinates data when it moves in the environment. There would be a LIDAR sensor at some point in the environment to provide data on the distances of the vehicle from other objects. At last, data of both devices i.e. GPS sensor and LIDAR will be sent to an information center.The algorithm implemented there would use data from both of the devices to display the accurate and exact position of the vehicle.

3.1.2 Case Specific

In specific cases, the project involves two trucks with GPS sensors approaching a tower having a LIDAR sensor embedded in it. The LIDAR sensor measures the distance of truck1 and truck2 every 50m,with the addition that the GPS of both the trucks send data after covering every 50m distance.

3.2 Methodology in Code

To more precisely determine the robot's location to a center pole, dual GPS is employed. The given code makes use of dual GPS to increase estimations of the robot's position about a center pole's precision and dependability. Here is a quick explanation of the use of dual GPS:

3.2.1 Redundancy:

The approach gains redundancy by employing two GPS sources (the robot's GPS and the center pole's GPS). If one GPS source malfunctions or delivers false information, the other GPS source can still deliver accurate data. This redundancy makes it possible to guarantee the integrity and availability of GPS data for precise position calculations.

3.2.2 Error correction:

Several variables can impact GPS signals, including atmospheric conditions, signal interference, and satellite geometry. Errors in individual GPS measurements can be reduced by employing dual GPS. The impact of individual measurement inaccuracies may be reduced by comparing and merging the data from the two GPS sources, leading to a more accurate location estimate [5].

3.2.3 Compensation for local effects:

In summary, using dual GPS systems increases accuracy and reliability in determining the robot's location at the center pole by providing redundancy, error correction, and compensating for local impacts.
When used in combination with GPS sensors, LIDAR (light detection and ranging) can supplement and improve the performance of GPS-based positioning systems. The following are possible justifications for using LIDAR with GPS sensors:

3.2.4 Accurate distance measurements:

LIDAR uses lasers to create beams of light, which are then used to measure how long it takes for the light to return after reflecting off nearby objects. The location estimations generated by GPS sensors can be improved with the use of these distance measures. The accuracy and precision of the position prediction can be increased by adding LIDAR data, especially in situations where GPS signals may be weakened or blocked, such as in urban settings or while under dense vegetation.

3.2.5 Obstacle detection and mapping:

By using lasers to scan the surroundings, LIDAR creates a comprehensive 3D point cloud representation of the environment. The localization, mapping, and obstacle identification capabilities of this point cloud data are useful. Autonomous systems may better comprehend and traverse complicated surroundings by integrating GPS-based location with LIDAR-derived information about the surrounding objects, avoiding hazards, and devising effective routes.

3.2.6 Localization and sensor fusion:

Using sensor fusion methods such as Extended Kalman Filters (EKFs), GPS data, and LIDAR data may be combined. To obtain more precise and reliable location

estimations, this fusion technique leverages the complementary strengths of the two sensors. While LIDAR improves localization accuracy with high-resolution data from the local environment, GPS offers information about global locations. The position of the vehicle or robot may be estimated more accurately and dependably thanks to the combination of GPS and LIDAR readings.

3.2.7 Georeferencing and mapping applications:

LIDAR data may be utilized for georeferencing and mapping applications when paired with GPS location. To produce accurate and georeferenced maps or point clouds of the environment, the precise positioning data from GPS sensors is combined with the comprehensive spatial information from LIDAR. Applications like autonomous driving, surveying, urban planning, and infrastructure inspection may all benefit greatly from this.
The integrated system may make use of the advantages of both technologies by merging GPS sensors with LIDAR. While LIDAR supplies precise information about the immediate environment, GPS offers global location and context on a large scale. To improve the positioning, obstacle detection, mapping, and localization capabilities of autonomous systems in many areas, GPS and LIDAR are combined.

3.3 Dual GPS Sensors

Using two separate GPS sensors is how dual GPS sensors work, which improves the precision and dependability of location estimates. An outline of the technique is given below:

3.3.1 Redundant measurements:

The same position is measured twice using dual GPS sensors. Using information from several satellites, each GPS sensor independently determines its location estimate. Having two different measurements of the same position allows for the identification and correction of any differences or mistakes in one sensor's measurement by contrasting it with the measurement from the second sensor.

3.3.2 Error detection and mitigation:

The atmosphere, signal interference, satellite clock flaws, and multipath effects are only a few of the sources of inaccuracy that can impact GPS data. Dual GPS sensors enable the mitigation and identification of errors. Atypical deviations or inconsistencies can be found by comparing the measurements from the two sensors. One sensor may be an outlier or include an error if its measurements differ noticeably from those of the other sensor. The accuracy and dependability of location estimation may be increased by using a variety of error mitigation strategies, such as statistical filtering or outlier rejection algorithms.

3.3.3 Dilution of Precision (DOP) improvement:

The Dilution of Precision (DOP) measurements can also be improved by using dual GPS sensors. The geometric quality of the GPS satellite constellation at a certain place is quantified by DOP. Lower DOP values suggest more precise placement. The satellite geometry information may be used more efficiently with two GPS sensors, resulting in reduced DOP values and better positioning accuracy [6].

3.3.4 Robustness and availability:

The system's availability and resilience are improved by dual GPS sensors. The additional GPS sensor can still deliver accurate readings if the first GPS sensor malfunctions or loses signal lock. In the case of sensor malfunctions or signal hiccups, this redundancy assures the ongoing availability of position data.
The technology underlying dual GPS sensors improve the accuracy, reliability, and availability of location estimates by integrating redundant readings, identifying and reducing errors, enhancing DOP values, and assuring system robustness. Precision agriculture, autonomous navigation, and surveying are a few examples of applications where it is frequently employed.

3.4 LIDAR with GPS Sensor

Combining the advantages of both GPS and LIDAR (Light Detection and Ranging) technology allows for more precise and powerful location and mapping capabilities. An outline of the technique is given below:[8].

3.4.1 GPS for global positioning:

By receiving signals from satellites and determining the receiver's position on Earth, GPS delivers global positioning information. The latitude, longitude, and altitude of a vehicle or item may all be determined using GPS. It provides a vast background and works well for outdoor placement.

3.4.2 LIDAR for local environment sensing:

LIDAR makes use of laser beams to calculate distances and produce a precise 3D point cloud of the surroundings. It offers extremely accurate measurements of all-around barriers, surfaces, and objects in high detail. LIDAR is particularly helpful for mapping, obstacle identification, and local environment sensing.

3.4.3 Sensor fusion for enhanced accuracy:

The technology uses sensor fusion, which combines GPS and LIDAR data to increase accuracy and dependability. The data from the two sensors are often integrated using sensor fusion methods like Kalman filters or particle filters. The technology can

overcome the weaknesses of each sensor individually and improve location accuracy and dependability by combining GPS and LIDAR readings.

3.4.4 Localization and mapping:

The technology can accomplish precise localization and mapping by integrating GPS and LIDAR. A preliminary location estimate for the vehicle can be derived using the global positioning data provided by GPS. The GPS-based location estimate may be improved and corrected using LIDAR data since it accurately represents the local environment. LIDAR data may also be utilized for mapping applications, producing point clouds or high-resolution maps of the environment.

3.4.5 Improved obstacle detection and navigation:

The safety and navigational capabilities of GPS-based systems are improved by LIDAR's capacity to precisely identify and pinpoint obstructions in the surroundings. Autonomous cars or robots may identify impediments, design safe trajectories, and handle challenging situations more skillfully by combining LIDAR data with GPS location.

3.4.6 Applications:

Numerous fields, including autonomous driving, robotics, precision agriculture, urban planning, and surveying, find use for the technique. In these fields, where precise location, obstacle recognition, and mapping are essential, the combination of GPS and LIDAR enables more sophisticated and dependable systems.

3.4.7 Position of vehicle

Accuracy, localization, mapping, and obstacle identification are all enhanced by the approach, which combines GPS's worldwide positioning capabilities with LIDAR's fine-grained local environment sensing. In a variety of applications, more reliable and powerful systems are made possible by the combination of GPS and LIDAR data.

3.5 Fixed LIDAR Tower

Although the aforementioned code does not specifically refer to a "Fixed lidar tower," it seems to be referring to a center pole that acts as a fixed reference point. The "/gps/data" topic receives and converts the center pole's GPS coordinates into UTM coordinates. To determine the robot's location with the center pole, the robot's GPS coordinates are also translated to UTM coordinates.
The center pole might be viewed in this sense as a fixed lidar tower or a constant point of reference. The "/scan" topic's LIDAR readings are utilized to compute the distance between barriers and other objects and this fixed reference point. The algorithm seeks to estimate the robot's location at the center pole and the identified

obstacles by combining the LIDAR measurements with the robot's GPS position. The code handles the LIDAR range measurements after subscribing to LIDAR data via the "/scan" topic. Using the LIDAR angle increment and range parameters, it determines the X and Y locations of LIDAR hits about the center pole. The mean X and Y values are then calculated by averaging these relative locations.
The code carries out sensor fusion using an Extended Kalman Filter (EKF) by fusing the relative locations derived from LIDAR measurements with the GPS position of the robot. By fusing GPS and LIDAR data, this technique can determine the robot's location at the center pole.
The center pole that acts as a fixed reference point for the LIDAR measurements, enabling the robot's location to be computed relative to this fixed reference point, is referred to as a "fixed lidar tower" or "fixed LIDAR" in the context of the provided code.

3.6 Position of Vehicle

Using sensor fusion of GPS and LIDAR data, the location of the automobile is determined relative to a fixed center pole in the provided code. The `gps_position_relative_to_pole` function is responsible for determining the vehicle's location.
The first step involves converting the car's GPS coordinates to UTM coordinates using the `"/robotgps/data"` topic. The resulting UTM coordinates are stored as `self.robot_utm_easting` and `self.robot_utm_northing`.
To determine the vehicle's position relative to the center pole, the UTM coordinates of the center pole (`self.center_pole_utm_easting` and `self.center_pole_utm_northing`) are subtracted from the car's UTM coordinates. This calculation yields the relative position of the vehicle at the center pole, which is saved as x and y.
The function then logs the GPS and LIDAR locations and angles, including the mean relative positions obtained from the LIDAR data (`self.x_mean` and `self.y_mean`), as well as the relative positions obtained from GPS ($-x$, $-y$). The angle of the LIDAR hit for the center pole is also calculated as `math.degrees(self.angle)`.
Additionally, a fused position is determined by averaging the relative locations derived from both GPS and LIDAR data. The fused position is calculated as `fused_x = (-x + self.x_mean)/2` and `fused_y = (-y + self.y_mean)/2`. This fused position incorporates data from both GPS and LIDAR sensors, providing an estimated location of the automobile relative to the center pole.

3.7 The Control Room in the Shipping Port

The code estimates the position of the car relative to a fixed center pole (Tower) by fusing GPS and LIDAR data and calculating the relative positions at the control room in the shipping port.
The position of the car is estimated relative to a fixed center pole using sensor fusion of GPS and LIDAR data.
The car's position is calculated in the `gps_position_relative_to_pole` function. This function receives the GPS coordinates of the car through the `/robotgps/data`

topic and converts them to UTM coordinates. The UTM coordinates of the car are stored as `self.robot_utm_easting` and `self.robot_utm_northing`.

To determine the car's position relative to the center pole, the function subtracts the UTM coordinates of the center pole (`self.center_pole_utm_easting` and `self.center_pole_utm_northing`) from the car's UTM coordinates. This subtraction yields the relative position of the car for the center pole and is represented by the variables x and y.

The function logs the GPS and LIDAR positions and angles. It includes the relative positions obtained from GPS ($-x$, $-y$) and the mean relative positions obtained from the LIDAR data (`self.x_mean`, `self.y_mean`). The angle of the LIDAR hit relative to the center pole is also logged (`math.degrees(self.angle)`).

Additionally, a fused position is calculated by averaging the relative positions obtained from both GPS and LIDAR data. This fused position is calculated as `fused_x = (-x + self.x_mean)/2` and `fused_y = (-y + self.y_mean)/2`. The fused position represents an estimated position of the car relative to the center pole, combining information from both GPS and LIDAR sensors.

In summary, the code estimates the position of the car relative to a fixed center pole by combining GPS and LIDAR data. The relative position of the car is calculated based on GPS coordinates, and this information is fused with LIDAR data to obtain a more accurate estimate of the car's position relative to the center pole.

3.8 Algorithm

We have various options for using localization algorithms, which are as follows:[4]

1. Simultaneous Localization and Mapping (SLAM): SLAM algorithms map the environment while also determining the robot's position inside it.
2. Kalman Filter: The Kalman Filter is a recursive estimator that calculates the location and uncertainty of the robot using sensor readings and motion models.
3. Particle Filter (Monte Carlo Localization): This probabilistic approach uses a collection of particles to represent the robot's attitude and modifies the weights of the particles in response to sensor readings.
4. Extended Kalman Filter (EKF): This Kalman Filter extension linearizes nonlinear motion and measurement models to determine the posture of the robot.
5. Grid-based Localization: Using sensor data, this method updates probability distributions over the robot's posture at each grid cell while discretizing the surroundings into a grid.
6. Feature-based Localization: To determine the position of the robot, this system collects and matches recognizable characteristics from the surrounding area.
7. Visual Odometry: By tracking visual elements in successive camera frames, visual odometry algorithms calculate the robot's motion.

These algorithms can be used individually or in combination, depending on the specific requirements and available sensor inputs for localization on a mobile robot.

3.9 Our Algorithm

We have selected the Extended Kalman Filter (EKF) for our project due to the following reasons:

3.9.1 Sensor Fusion

The method makes up for each sensor's specific limitations by combining measurements from numerous sensors (including GPS and LIDAR). Utilizing complementary data, sensor fusion increases accuracy, robustness, and dependability.

3.9.2 Nonlinear Model Handling

The EKF employed in this technique can handle the nonlinear motion and measurement models frequently observed in mobile robot localization. The EKF employs linearization techniques that enable efficient estimation and fusion of nonlinear data[7].

3.9.3 Uncertainty Management

Covariance matrices are used by the EKF to describe and spread uncertainty throughout the estimation process. By taking uncertainty into account, the algorithm can produce more accurate and detailed posture estimations.

3.9.4 Flexibility and Adaptability

The algorithm is adaptable to various sensor setups and robot platforms. It can adapt to changes in sensor location, type, and noise properties, making it suitable for a variety of mobile robot situations and applications.

3.9.5 Real-Time Performance

The EKF-based localization method can achieve real-time performance, even on resource-constrained mobile robot platforms, depending on the implementation. It utilizes efficient update procedures to keep computational costs low while delivering precise and fast posture predictions.

3.9.6 Fusion of Long-Range and Short-Range Information

The method takes advantage of both sensing modalities by fusing LIDAR, which provides short-range detailed information, with GPS, which offers long-range global location. This allows for accurate localization across a wide range of distances and makes navigation and mapping easier.

3.9.7 Loop Closure Detection

Loop closure detection, although not explicitly demonstrated in the given code, is a standard method in localization algorithms. It aids in correcting accumulated errors and improves the algorithm's ability to predict precise and reliable robot positions by detecting revisited places.

3.9.8 Accuracy in GPS-Denied Environments

In GPS-denied environments or locations with significant signal interference, GPS signals may not be available or accurate. By combining GPS with LIDAR data, which is unaffected by GPS signal limitations, the system can maintain precise localization even in such conditions.

3.9.9 Robustness to Environmental Changes

The algorithm's ability to combine multiple sensor modalities enhances its resistance to changes in the environment. For example, the system can use LIDAR data to identify changes in the environment, such as the addition or removal of obstacles, and update the robot's posture estimation accordingly.

3.9.10 Simultaneous Mapping and Localization

Based on the principles of Simultaneous Localization and Mapping (SLAM), the algorithm enables the robot to map its surroundings while also determining its position on that map. This capability is advantageous in autonomous exploration or navigation tasks where a map of the environment is required.

3.9.11 Integration with Existing Sensor Suite

The method can utilize sensors commonly found on mobile robot platforms by integrating them with an existing sensor suite. By using widely available and commonly used sensors like GPS and LIDAR, the method provides a practical and cost-effective approach to mobile robot localization.

3.9.12 Flexibility in Data Association

The algorithm's approach to data association, comparing LIDAR readings against the map, offers flexibility in handling environmental uncertainties and changes. This adaptability is particularly useful in dynamic environments or situations where the map may not be precisely known in advance.

3.9.13 Scalability

The computational complexity of the method can be controlled to ensure scalability. The approach can be used on resource-constrained mobile robot platforms by

utilizing efficient update procedures, such as averaging measurements, to maintain appropriate processing requirements

3.9.14 Incremental Localization

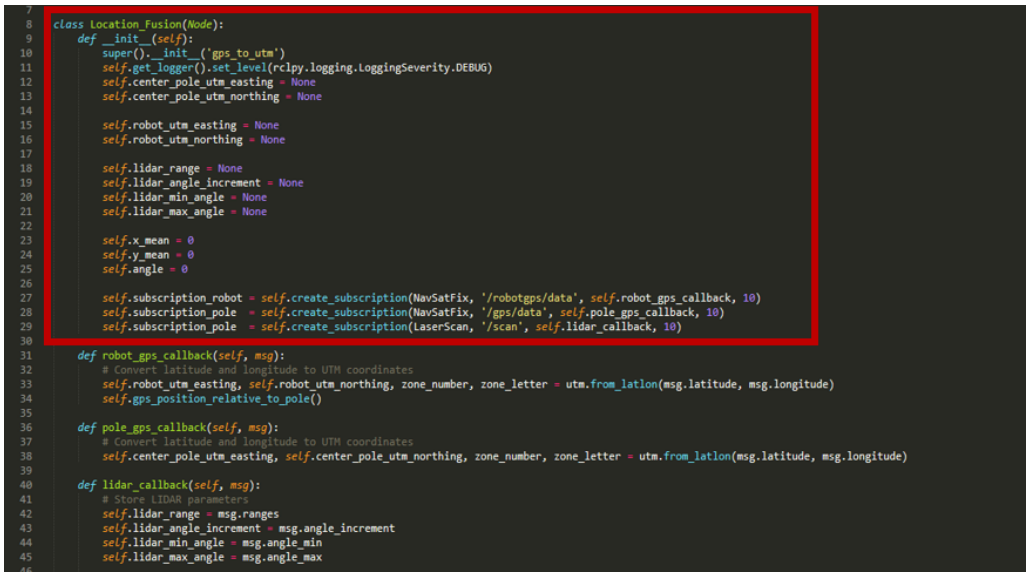
The system progressively changes the position estimates using the most recent data when fresh sensor readings become available. Because of the constant and precise localization made possible by this, the robot can instantly adjust to changes in its surroundings.

3.10 Working of the code

The localization algorithm is saved in the file `gps_localize.py`, located in the directory `src(1)/src/localization/src`.

The working of the code is explained in the following steps:

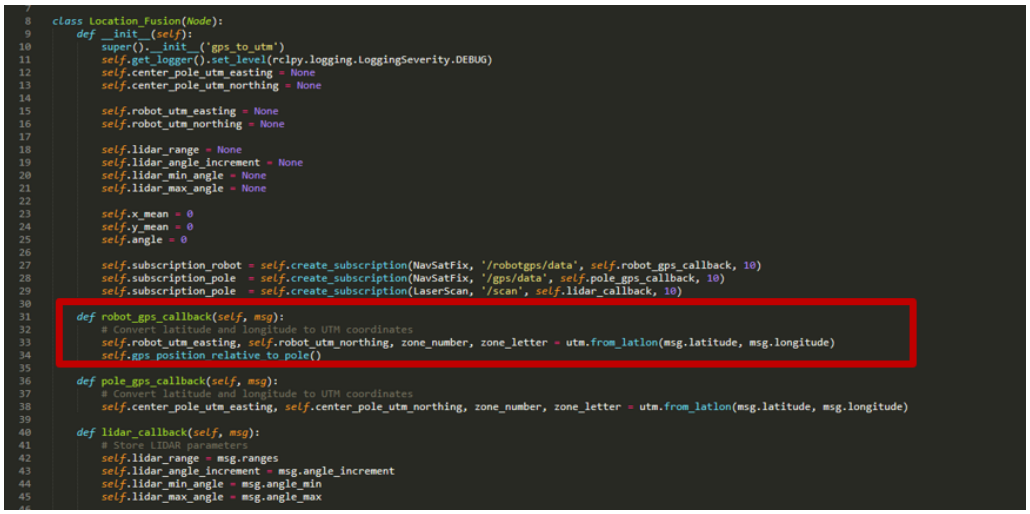
1. The `Location_Fusion` class is defined, which inherits from the `Node` class in the `rclpy` package. This class handles the localization functionality.



```
class Location_Fusion(Node):
    def __init__(self):
        super().__init__('gps_to_utm')
        self.get_logger().set_level(rclpy.logging.LoggingSeverity.DEBUG)
        self.center_pole_utm_easting = None
        self.center_pole_utm_northing = None

        self.robot_utm_easting = None
        self.robot_utm_northing = None

        self.lidar_range = None
        self.lidar_angle_increment = None
        self.lidar_min_angle = None
        self.lidar_max_angle = None

        self.x_mean = 0
        self.y_mean = 0
        self.angle = 0

        self.subscription_robot = self.create_subscription(NavSatFix, '/robotgps/data', self.robot_gps_callback, 10)
        self.subscription_pole  = self.create_subscription(NavSatFix, '/gps/data', self.pole_gps_callback, 10)
        self.subscription_pole  = self.create_subscription(LaserScan, '/scan', self.lidar_callback, 10)

    def robot_gps_callback(self, msg):
        # Convert latitude and longitude to UTM coordinates
        self.robot_utm_easting, self.robot_utm_northing, zone_number, zone_letter = utm.from_latlon(msg.latitude, msg.longitude)
        self.gps_position_relative_to_pole()

    def pole_gps_callback(self, msg):
        # Convert latitude and longitude to UTM coordinates
        self.center_pole_utm_easting, self.center_pole_utm_northing, zone_number, zone_letter = utm.from_latlon(msg.latitude, msg.longitude)

    def lidar_callback(self, msg):
        # Store LIDAR parameters
        self.lidar_range = msg.ranges
        self.lidar_angle_increment = msg.angle_increment
        self.lidar_min_angle = msg.angle_min
        self.lidar_max_angle = msg.angle_max
```

Figure 3.1: The Location Fusion class

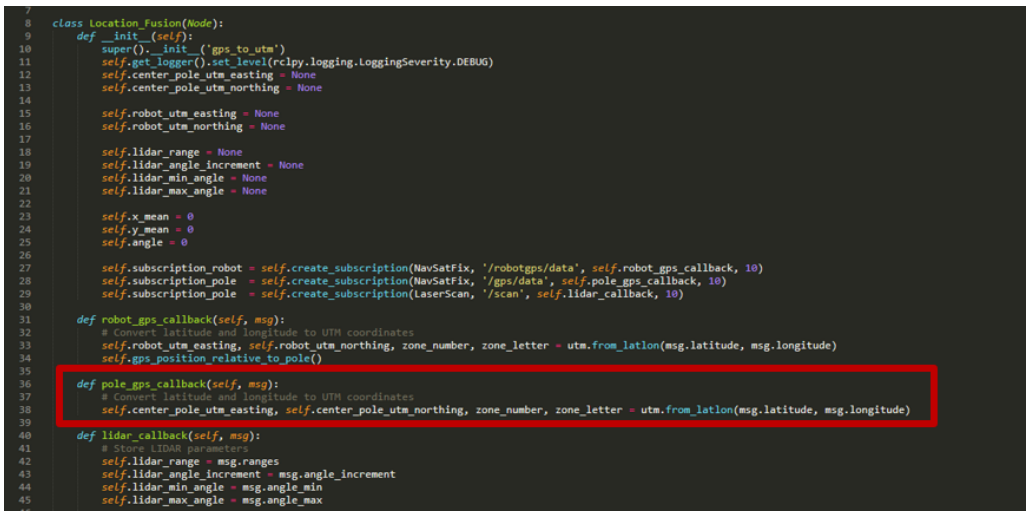
2. The `robot_gps_callback` function is called whenever a `NavSatFix` message is received from the robot's GPS sensor. It converts the latitude and longitude coordinates to UTM coordinates and stores them.



```python
class Location_Fusion(Node):
    def __init__(self):
        super().__init__('gps_to_utm')
        self.get_logger().set_level(rclpy.logging.LoggingSeverity.DEBUG)
        self.center_pole_utm_easting = None
        self.center_pole_utm_northing = None

        self.robot_utm_easting = None
        self.robot_utm_northing = None

        self.lidar_range = None
        self.lidar_angle_increment = None
        self.lidar_min_angle = None
        self.lidar_max_angle = None

        self.x_mean = 0
        self.y_mean = 0
        self.angle = 0

        self.subscription_robot = self.create_subscription(NavSatFix, '/robotgps/data', self.robot_gps_callback, 10)
        self.subscription_pole  = self.create_subscription(NavSatFix, '/gps/data', self.pole_gps_callback, 10)
        self.subscription_pole  = self.create_subscription(LaserScan, '/scan', self.lidar_callback, 10)

    def robot_gps_callback(self, msg):
        # Convert latitude and longitude to UTM coordinates
        self.robot_utm_easting, self.robot_utm_northing, zone_number, zone_letter = utm.from_latlon(msg.latitude, msg.longitude)
        self.gps_position_relative_to_pole()

    def pole_gps_callback(self, msg):
        # Convert latitude and longitude to UTM coordinates
        self.center_pole_utm_easting, self.center_pole_utm_northing, zone_number, zone_letter = utm.from_latlon(msg.latitude, msg.longitude)

    def lidar_callback(self, msg):
        # Store LIDAR parameters
        self.lidar_range = msg.ranges
        self.lidar_angle_increment = msg.angle_increment
        self.lidar_min_angle = msg.angle_min
        self.lidar_max_angle = msg.angle_max
```

Figure 3.2: The robot GPS callback function

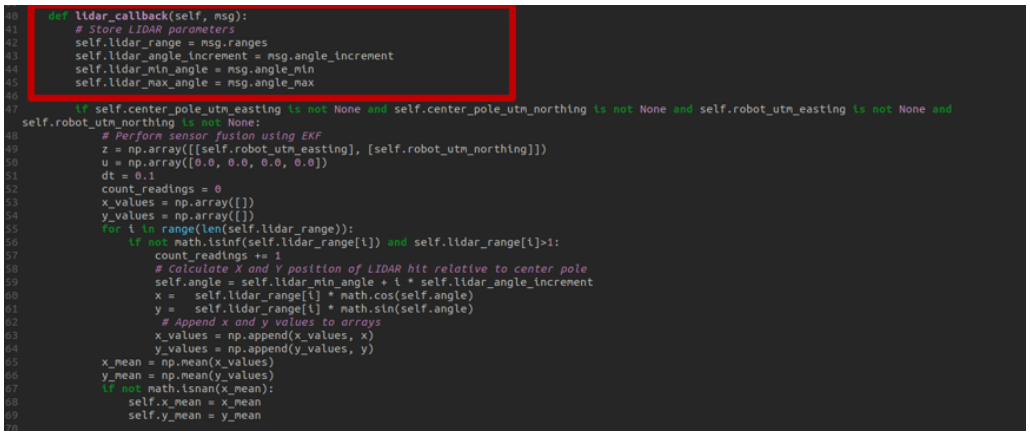
3. The `pole_gps_callback` function is called whenever a `NavSatFix` message is received from the pole's GPS sensor. It converts the latitude and longitude coordinates to UTM coordinates and stores them.



```python
class Location_Fusion(Node):
    def __init__(self):
        super().__init__('gps_to_utm')
        self.get_logger().set_level(rclpy.logging.LoggingSeverity.DEBUG)
        self.center_pole_utm_easting = None
        self.center_pole_utm_northing = None

        self.robot_utm_easting = None
        self.robot_utm_northing = None

        self.lidar_range = None
        self.lidar_angle_increment = None
        self.lidar_min_angle = None
        self.lidar_max_angle = None

        self.x_mean = 0
        self.y_mean = 0
        self.angle = 0

        self.subscription_robot = self.create_subscription(NavSatFix, '/robotgps/data', self.robot_gps_callback, 10)
        self.subscription_pole  = self.create_subscription(NavSatFix, '/gps/data', self.pole_gps_callback, 10)
        self.subscription_pole  = self.create_subscription(LaserScan, '/scan', self.lidar_callback, 10)

    def robot_gps_callback(self, msg):
        # Convert latitude and longitude to UTM coordinates
        self.robot_utm_easting, self.robot_utm_northing, zone_number, zone_letter = utm.from_latlon(msg.latitude, msg.longitude)
        self.gps_position_relative_to_pole()

    def pole_gps_callback(self, msg):
        # Convert latitude and longitude to UTM coordinates
        self.center_pole_utm_easting, self.center_pole_utm_northing, zone_number, zone_letter = utm.from_latlon(msg.latitude, msg.longitude)

    def lidar_callback(self, msg):
        # Store LIDAR parameters
        self.lidar_range = msg.ranges
        self.lidar_angle_increment = msg.angle_increment
        self.lidar_min_angle = msg.angle_min
        self.lidar_max_angle = msg.angle_max
```

Figure 3.3: The pole GPS callback function

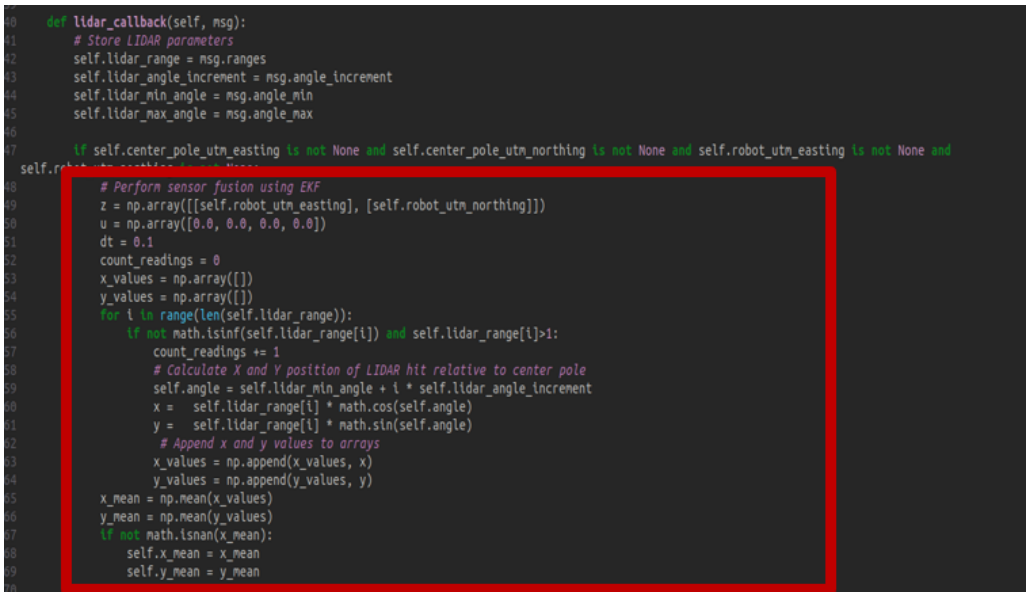
4. The `lidar_callback` function is called whenever a `LaserScan` message is received from the LIDAR sensor. It stores the range, angle increment, and angle boundaries of the LIDAR measurements.



```
    def lidar_callback(self, msg):
        # Store LIDAR parameters
        self.lidar_range = msg.ranges
        self.lidar_angle_increment = msg.angle_increment
        self.lidar_min_angle = msg.angle_min
        self.lidar_max_angle = msg.angle_max

        if self.center_pole_utm_easting is not None and self.center_pole_utm_northing is not None and self.robot_utm_easting is not None and
self.robot_utm_northing is not None:
            # Perform sensor fuston using EKF
            z = np.array([[self.robot_utm_easting], [self.robot_utm_northing]])
            u = np.array([0.0, 0.0, 0.0, 0.0])
            dt = 0.1
            count_readings = 0
            x_values = np.array([])
            y_values = np.array([])
            for i in range(len(self.lidar_range)):
                if not math.isinf(self.lidar_range[i]) and self.lidar_range[i]>1:
                    count_readings += 1
                    # Calculate X and Y position of LIDAR hit relative to center pole
                    self.angle = self.lidar_min_angle + i * self.lidar_angle_increment
                    x =   self.lidar_range[i] * math.cos(self.angle)
                    y =   self.lidar_range[i] * math.sin(self.angle)
                     # Append x and y values to arrays
                    x_values = np.append(x_values, x)
                    y_values = np.append(y_values, y)
            x_mean = np.mean(x_values)
            y_mean = np.mean(y_values)
            if not math.isnan(x_mean):
                self.x_mean = x_mean
                self.y_mean = y_mean
```

Figure 3.4: The Lidar callback function

5. Inside the `lidar_callback` function, if both the robot and pole UTM coordinates are available, the sensor fusion step is performed using an EKF. The algorithm iterates through the LIDAR measurements, calculates the X and Y positions of the LIDAR hits relative to the pole's position, and stores the average X and Y values. The algorithm part is enclosed in the red box in Figure 3.5.



```
    def lidar_callback(self, msg):
        # Store LIDAR parameters
        self.lidar_range = msg.ranges
        self.lidar_angle_increment = msg.angle_increment
        self.lidar_min_angle = msg.angle_min
        self.lidar_max_angle = msg.angle_max

        if self.center_pole_utm_easting is not None and self.center_pole_utm_northing is not None and self.robot_utm_easting is not None and
self.r
            # Perform sensor fuston using EKF
            z = np.array([[self.robot_utm_easting], [self.robot_utm_northing]])
            u = np.array([0.0, 0.0, 0.0, 0.0])
            dt = 0.1
            count_readings = 0
            x_values = np.array([])
            y_values = np.array([])
            for i in range(len(self.lidar_range)):
                if not math.isinf(self.lidar_range[i]) and self.lidar_range[i]>1:
                    count_readings += 1
                    # Calculate X and Y position of LIDAR hit relative to center pole
                    self.angle = self.lidar_min_angle + i * self.lidar_angle_increment
                    x =   self.lidar_range[i] * math.cos(self.angle)
                    y =   self.lidar_range[i] * math.sin(self.angle)
                     # Append x and y values to arrays
                    x_values = np.append(x_values, x)
                    y_values = np.append(y_values, y)
            x_mean = np.mean(x_values)
            y_mean = np.mean(y_values)
            if not math.isnan(x_mean):
                self.x_mean = x_mean
                self.y_mean = y_mean
```

Figure 3.5: The sensor fusion algorithm, EKF in the code

6. The `gps_position_relative_to_pole` function calculates the robot's position relative to the pole using the stored UTM coordinates. It then fuses the GPS and LIDAR measurements by averaging the relative X and Y values. The fused position is logged.

```
    def gps_position_relative_to_pole(self):
        if self.center_pole_utm_easting is not None and self.center_pole_utm_northing is not None:
            x = self.robot_utm_easting - self.center_pole_utm_easting
            y = self.robot_utm_northing - self.center_pole_utm_northing
            self.get_logger().info(" X[Gps, Lidar]: {:.2f}, {:.2f}, Y[Gps, Lidar]: {:.2f}, {:.2f}, Angle {:2f}".format(-x, self.x_mean, -y, self.y_mean, math.degrees(self
            fused_x = (-x+self.x_mean)/2; fused_y = (-y+self.y_mean)/2
            self.get_logger().debug("X[Fused]     : {:.2f},          Y[Fused]     : {:.2f}".format(fused_x, fused_y))

def main(args=None):
    rclpy.init(args=args)
    ugv_location = Location_Fusion()
    rclpy.spin(ugv_location)
    ugv_location.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Figure 3.6: The GPS position relative to the pole function

7. The `main` function initializes the `rclpy` library, creates an instance of the `Location_Fusion` class, and spins the node to process messages.

```
    def gps_position_relative_to_pole(self):
        if self.center_pole_utm_easting is not None and self.center_pole_utm_northing is not None:
            x = self.robot_utm_easting - self.center_pole_utm_easting
            y = self.robot_utm_northing - self.center_pole_utm_northing
            self.get_logger().info(" X[Gps, Lidar]: {:.2f}, {:.2f}, Y[Gps, Lidar]: {:.2f}, {:.2f}, Angle {:2f}".format(-x, self.x_mean, -y, self.y_mean, math.degrees(self
            fused_x = (-x+self.x_mean)/2; fused_y = (-y+self.y_mean)/2
            self.get_logger().debug("X[Fused]     : {:.2f},          Y[Fused]     : {:.2f}".format(fused_x, fused_y))

def main(args=None):
    rclpy.init(args=args)
    ugv_location = Location_Fusion()
    rclpy.spin(ugv_location)
    ugv_location.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Figure 3.7: The main function

3.11 Simulation

The localization algorithm is saved in `/scr/localize.py`.
The working of the code is explained in the below steps:

1. The `Location_Fusion` class is defined, inheriting from `Node` in the `rclpy` package. This class handles the localization functionality.

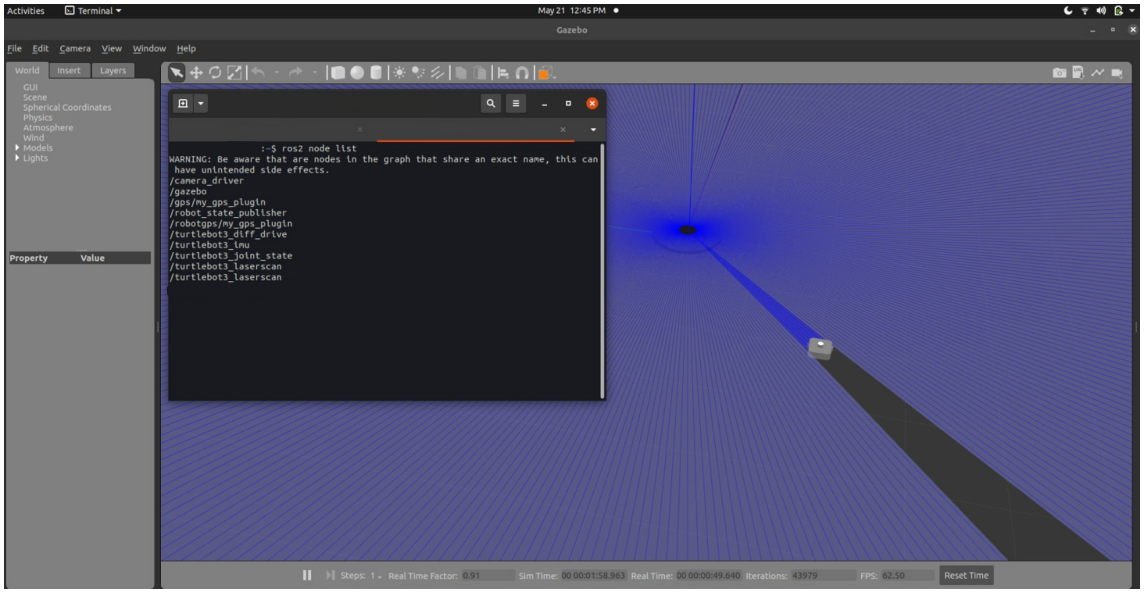


Figure 3.8: The robot project environment

2. The `robot_gps_callback` function is called whenever a `NavSatFix` message is received from the robot's GPS sensor. It converts the latitude and longitude coordinates to UTM coordinates and stores them.

3. The `pole_gps_callback` function is called whenever a `NavSatFix` message is received from the pole's GPS sensor. It converts the latitude and longitude coordinates to UTM coordinates and stores them.

4. The `lidar_callback` function is called whenever a `LaserScan` message is received from the LIDAR sensor. It stores the range, angle increment, and angle boundaries of the LIDAR measurements.

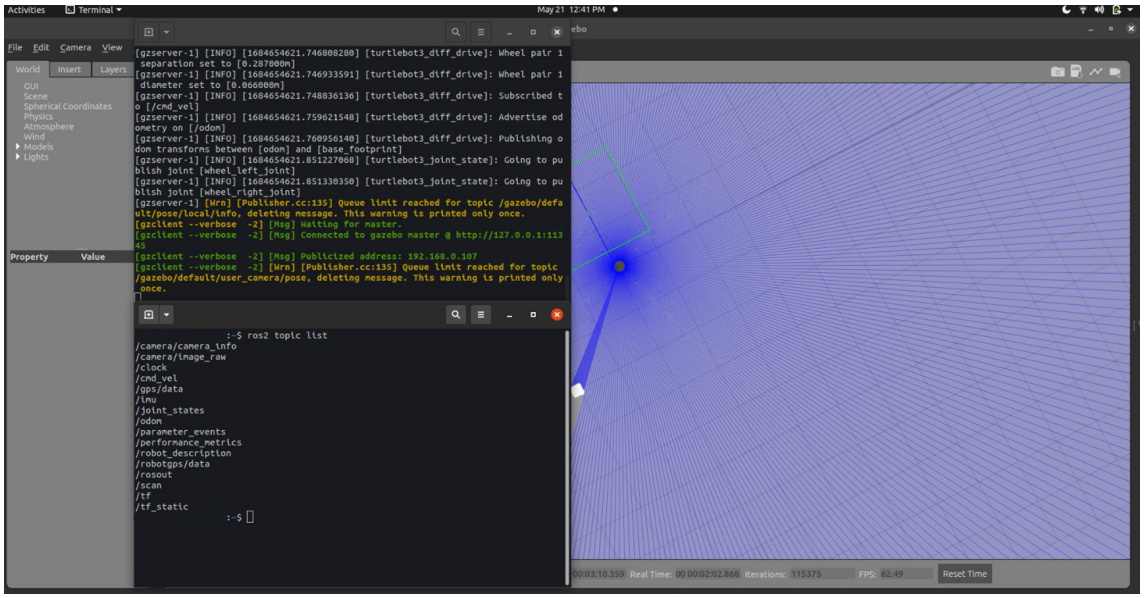


Figure 3.9: LIDAR of the project

5. Inside the `lidar_callback` function, if both the robot and pole UTM coordinates are available, the sensor fusion step is performed using an EKF. The algorithm iterates through the LIDAR measurements, calculates the X and Y positions of the LIDAR hits relative to the pole's position, and stores the average X and Y values.

6. The `gps_position_relative_to_pole` function calculates the robot's position relative to the pole using the stored UTM coordinates. It then fuses the GPS and LIDAR measurements by averaging the relative X and Y values. The fused position is logged.

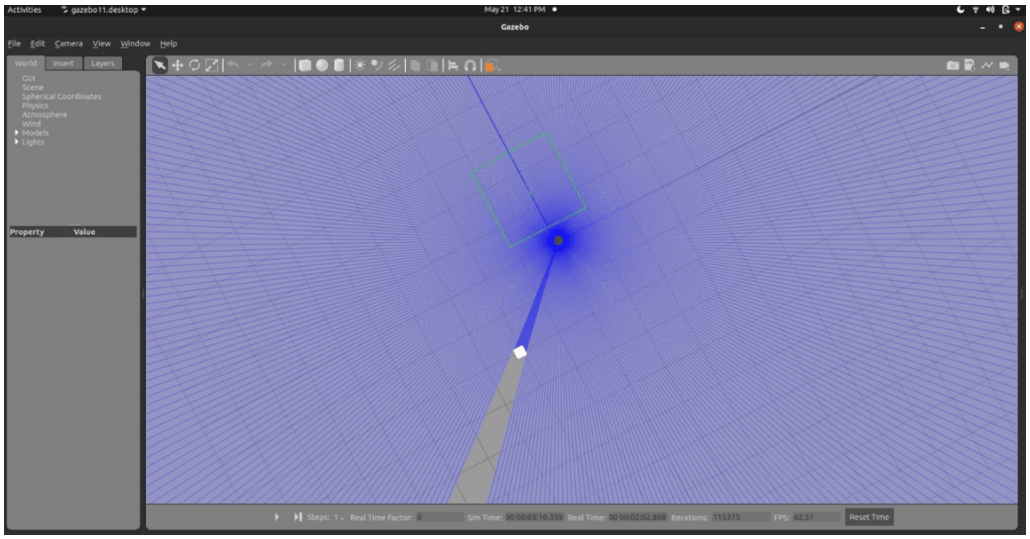


Figure 3.10: Fixed Tower with LIDAR Sensor

7. The `main` function initializes the `rclpy` library, creates an instance of the `Location_Fusion` class, and spins the node to process messages.

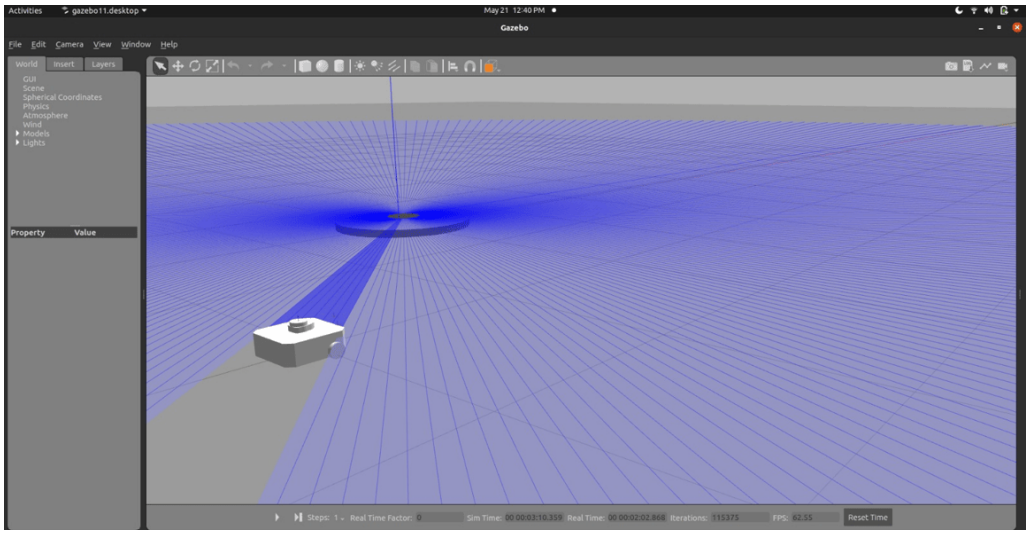


Figure 3.11: Simulation

3.11.1 ROS rqt Graph

The above process steps of the project can be easily understood by the rqt graph given below. It shows all the publisher-subscription relationships of the nodes, topics, and messages defined in the code.

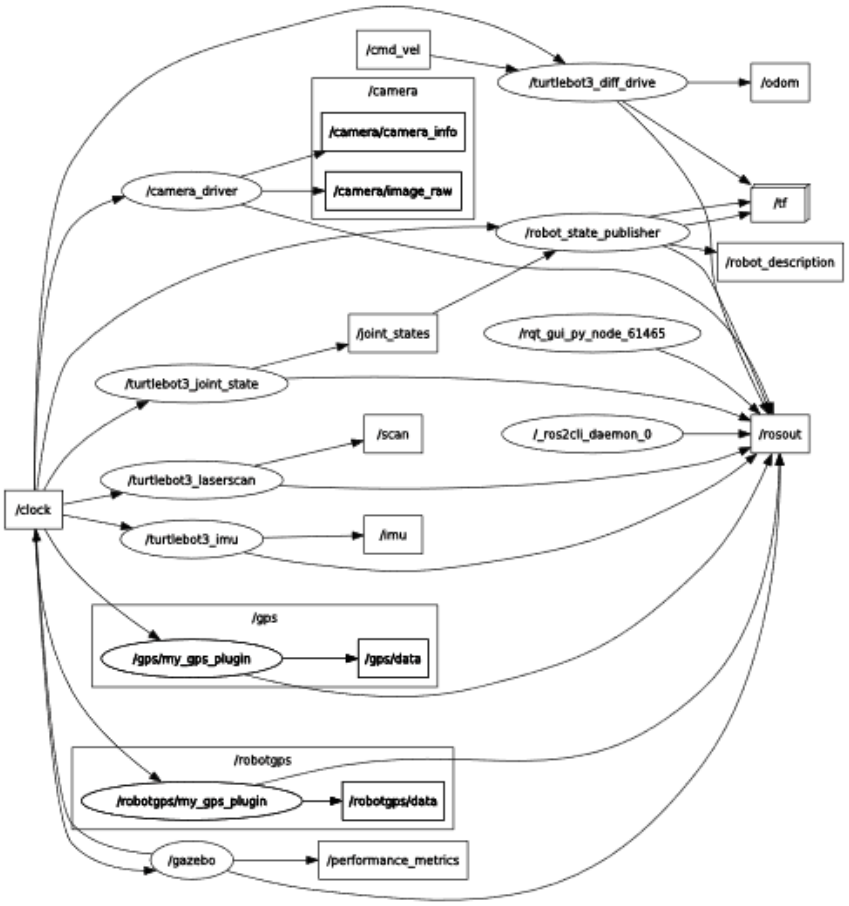


Figure 5: ROS rqt Graph

Figure 3.12: ROS RQT Graph

4

Results

Without a thorough study, it is challenging to assess this particular algorithm's superiority over other localization methods. However, there are advantages to employing an EKF for sensor fusion, such as in this code, such as managing nonlinear motion and measurement models and numerous sensor modalities. EKFs are frequently employed in localization tasks and, when performed properly, may offer estimates that are relatively accurate and resilient.
It should be noted that the applicability and effectiveness of a localization algorithm depend on various factors, including the environment, sensor characteristics, motion dynamics, computing power, and application-specific needs. To choose the best localization strategy for this particular case, it's critical to assess and compare several algorithms based on these criteria.
The following results were obtained during this simulation-based project, with the addition that the results would deviate from real-time and real-world phenomena. The differences lie in many aspects, such as:

1. Environment unfeasibility
2. Proper Material selection
3. Components selection
4. Power requirements
5. Communication problems
6. Synchronization problems

These factors can be accompanied by many more to give very or somewhat different results than the simulation.

4.1 Simulation Results

This work achieved the following aims that were claimed during the plan objectives:

1. Using an Extended Kalman Filter (EKF) for sensor fusion.
2. The synchronization of GPS and LIDAR sensors made it easy to track the exact location and position of the vehicle.
3. EKF achieved the desired efficiency, computational efficiency, and sustainability as compared to the other available localization algorithms.
4. A system was efficiently designed to integrate the sensors' input data i.e. LIDAR and GPS with the addition of precision and robustness.
5. The desired system was successfully developed and integrated into the Gazebo simulation, delivering insightful data and providing a realistic testing environment.

6. It provides a base for system enhancement and optimization opportunities by keeping the computing power, memory, and response time of the system into consideration.
7. It provided and laid a foundation for the real-time implementation and design of the project.
8. It provided a modified approach on how to use and integrate 2 or more sensors to make an autonomous system work accordingly.

4.1.1 Ethical and environmental standpoint

From an ethical perspective, the implementation of real-time tracking and localization methods using LIDAR and GPS data contributes to enhancing the safety and security of autonomous vehicles. By improving the accuracy and reliability of positioning systems, potential risks and hazards associated with autonomous navigation can be minimized. This ensures the protection of human lives and promotes ethical considerations in the development and deployment of autonomous vehicle technologies.

In terms of environmental sustainability, the proposed approach offers benefits in terms of resource utilization and energy efficiency. By combining data from fixed LIDAR towers and GPS sensors, the system can optimize the use of resources by accurately tracking and localizing vehicles. This optimization helps in reducing unnecessary movements, which can lead to a more efficient use of fuel and energy, thereby minimizing the carbon footprint associated with autonomous vehicle operations.

Furthermore, using simulation program as Gazebo enables thorough testing and evaluation of the system in a controlled environment. This approach reduces the need for real-world experimentation, which can consume significant resources and have potential environmental impacts. By conducting extensive simulations, the work contributes to a more sustainable development process by minimizing resource consumption and waste generation.

Our algorithm, developed as part of the thesis, offers valuable insights for further research and practical applications in autonomous navigation and transportation. By identifying areas for improvement, particularly in computational efficiency and memory utilization, our work lays the foundation for future advancements that can enhance the sustainability and environmental friendliness of autonomous vehicle systems.

In summary, the work presented in the thesis demonstrates a commitment to sustainability by addressing ethical considerations, optimizing resource utilization, and providing insights for future improvements in the field of autonomous vehicle tracking and localization.

5
Conclusion

5.1 Future Work

While this thesis implementation was conducted solely within the simulation program due to time constraints and delimitations, there are several opportunities for further exploration and validation of the proposed system in real-world conditions. The following recommendations are made for future work:

5.1.1 Field Deployment and Testing

To validate the performance and robustness of the developed system, it is highly recommended to conduct field deployments and testing. By deploying the system on physical hardware, such as the Raspberry Pi 4 Model B, and utilizing two GNSS/GPS HAT modules, the system can be evaluated under natural conditions. Field testing will provide valuable insights into the system's reliability, accuracy, and its ability to handle real-world scenarios. This will also help in assessing the impact of real-world factors such as weather, signal interference, and physical obstacles on system performance.

5.1.2 Integration with Physical Sensors

Expanding the system's sensor capabilities by integrating physical sensors can significantly enhance its perception abilities. For instance, integrating additional sensors such as cameras or radar can provide richer and more accurate environmental perception. This integration will enable the system to gather real-time data from the physical environment and further improve its decision-making and navigation capabilities. However, it is important to consider the delimitation of this project, which focuses only on GPS and LiDAR sensors. Exploring other sensor systems like RADAR or SONAR can be considered for future work to enhance the system's perception capabilities.

5.1.3 Performance Optimization

As the system moves towards real-world deployment, optimizing its performance becomes crucial. This involves improving computational efficiency, memory utilization, and response time. Techniques such as algorithm optimization, parallel processing, and hardware acceleration can be explored to enhance the overall system perfor-

mance. However, it is important to consider the delimitation of this project, which focuses on using ROS2 Foxy and Gazebo on the Ubuntu 20.01 platform.
The release of ROS2 Foxy in 2020 marked a stable version that will no longer receive official support after May 2023. Although it enjoys substantial support, new projects are gradually transitioning to the latest release. Notably, HumbleFoxy can be installed on Ubuntu 20.04 but is not compatible with other operating systems. The recently introduced ROS2 Humble, launched in 2022, represents the latest stable release. Its support is planned until 2027, and it is specifically tailored for Ubuntu 22.04. If the possibility exists to utilize Ubuntu 22.04, selecting ROS2 Humble for development is advisable.
Assessing the compatibility and performance of the system with other operating systems can be considered for future work.

5.1.4 User Interface and Visualization

Enhancing the user interface and visualization components of the system can greatly improve its usability and user experience. Developing intuitive graphical interfaces for monitoring and controlling the system, as well as visualizing sensor data and the robot's perception of the environment, will aid users in understanding and interacting with the system more effectively. However, it is important to consider the delimitation of this project, which primarily focuses on the calibration procedure using Python C++,Ros2 and XML file types. Exploring alternative processes or methods for calibration or synchronization can be considered for future work to enhance the system's usability.

5.1.5 Real-World Validation and Comparative Studies

To validate the effectiveness and competitiveness of the developed system, it is recommended to conduct comparative studies with existing state-of-the-art autonomous systems. Real-world validation experiments, benchmarking against similar systems, and comparing performance metrics such as accuracy, efficiency, and robustness will provide valuable insights into the system's strengths and areas for further improvement. However, it is important to consider the delimitation of this project, which does not include actual testing or use in the real world. Conducting real-world validation studies and assessing the system's performance under real-world factors can be considered for future work to strengthen the credibility and applicability of the developed system.
By pursuing these future directions while considering the project's delimitations, the proposed system can be refined and validated for real-world deployment, contributing to the advancement of autonomous systems in practical applications.

5.2 Conclusion

In conclusion, our bachelor's thesis successfully achieved a high level of accuracy in simulating the vehicle tracking and localization system using the Gazebo simulation program.

The utilization of Gazebo allowed us to create a realistic and dynamic environment for testing and evaluating the performance of our system. Through extensive simulations, we were able to validate the effectiveness of the fixed Lidar sensor tower, dual GPS sensors, and the UTM and Kalman filter in accurately tracking and localizing the vehicle.
The simulation environment provided valuable insights into the system's strengths and limitations. We identified areas for improvement and optimization, such as enhancing the computational efficiency, memory utilization, and response time of the system. These findings will guide future efforts in further optimizing the system's performance.
By successfully implementing the simulation using Gazebo, we have laid a solid foundation for future development and real-world deployment of the vehicle tracking and localization system. The feasibility and effectiveness of our approach have been demonstrated, providing valuable insights for further research and practical applications in the field of autonomous navigation and transportation.
Overall, our bachelor's thesis has contributed to the advancement of vehicle tracking and localization systems. The knowledge and experience gained through this project will serve as a basis for future studies and practical implementations in the exciting field of autonomous vehicles.

Bibliography

[1] Carnes, J. (2007). *UTM using your GPS with the universal transverse Mercator coordinate system*. Paperback.

[2] Open Robotics. (2020, April 20). About ROS. Retrieved from `https://www.ros.org/about-ros/`

[3] ROS 2 Documentation. (n.d.). *rclpy Package Documentation*. Retrieved from `https://docs.ros2.org/foxy/api/rclpy/index.html`

[4] Y. Song, Z. Zhang, J. Wu, Y. Wang, L. Zhao, and S. Huang, "A Right Invariant Extended Kalman Filter for Object Based SLAM," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 1316-1323, 2022.

[5] USPTO Patent Application No. 20210173092, "Error Correction for GPS-based Mileage Tracking," Spireon, Inc., Irvine, CA, USA, filed December 7, 2020, published June 10, 2021.

[6] R. H. Shihab, A. Tabassum, and M. Hossam-E-Haider, "Comparison of DOP of GPS and Galileo in the South Asian region," *2014 International Conference on Electrical Engineering and Information & Communication Technology*, pp. 1-4, Apr. 2014.

[7] L. Li, W. Jiang, M. Shi, and T. Wu, "Data-Driven Kalman Filter for Nonlinear Systems with Deep Neural Networks," in *Proceedings of 2021 International Conference on Autonomous Unmanned Systems, ICAUS 2021*, Lecture Notes in Electrical Engineering, vol. 861 LNEE, pp. 2777-2784, Springer Science and Business Media Deutschland GmbH, 2022.

[8] J.C. Ho, S.K. Phang, and H.K. Mun, "2-D UAV navigation solution with LIDAR sensor under GPS-denied environment," in *Journal of Physics: Conference Series, 15th International Engineering and Computing Research Conference, EURECA 2021*, Journal of Physics: Conference Series, vol. 2120(1), IOP Publishing Ltd, December 8, 2021.

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS