UNIVERSITY
OF TURKU

Image-guided Landmark-based Localization and Mapping with LiDAR

Robotics and autonomous systems

Master’s Degree Programme in Information and Communication Technology

Department of Computing, Faculty of Technology

Master of Science in Technology Thesis

Author:

Alberto Carlos Hernandez Ledesma

Supervisor(s):

MSc (Tech) Qingqing Li

Prof. Tomi Westerlund

March 2023

Master of Science in Technology Thesis

Department of Computing, Faculty of Technology
University of Turku

Subject: Robotics and autonomous systems
Programme: Master's Degree Programme in Information and Communication Technology
Author: Alberto Carlos Hernandez Ledesma
Title: Image-guided Landmark-based Localization and Mapping with LiDAR
Supervisor(s): Msc (Tech) Qingqing Li, Prof. Tomi Westerlund
Number of pages: 55
Date: March 2023

Abstract.

Mobile robots must be able to determine their position to operate effectively in diverse environments. The presented work proposes a system that integrates LiDAR and camera sensors and utilizes the YOLO object detection model to identify objects in the robot's surroundings. The system, developed in ROS, groups detected objects into triangles, utilizing them as landmarks to determine the robot's position. A triangulation algorithm is employed to obtain the robot's position, which generates a set of nonlinear equations that are solved using the Levenberg-Marquardt algorithm.

The presented work comprehensively discusses the proposed system's study, design, and implementation. The investigation begins with an overview of current SLAM techniques. Next, the system design considers the requirements for localization and mapping tasks and an analysis comparing the proposed approach to the contemporary SLAM methods. Finally, we evaluate the system's effectiveness and accuracy through experimentation in the Gazebo simulation environment, which allows for controlling various disturbances that a real scenario can introduce.

Keywords: Gazebo, Image-guided landmark-based localization, LiDAR, ROS 2, SLAM, YOLO

Contents

List of Acronyms

AGV	Automated Guided Vehicle
API	Application Programming Interface
AV	Autonomous Vehicle
FoV	Field of View
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
ICP	Iterative Closes Point
INS	Inertial Navigation System
LiDAR	Light Detection And Ranging
RGB	Red Green Blue
ROS	Robot Operating System
SDF	Simulation Description Format
SLAM	Simultaneous Localization and Mapping
UAV	Unmanned Aerial Vehicle
URDF	Unified Robot Description Format
V-SLAM	Visual-Simultaneous Localization and Mapping
XML	Extensible Markup Language
YOLO	You Only Look Once

1 Introduction

In recent years, the field of robotics has made significant progress, and one of the biggest challenges in this area is the localization of robots. Precise localization is essential [1] for a robot to perform tasks effectively and navigate its environment. Researchers have proposed various approaches to address this challenge, including landmarks localization and mapping [2,3].

One technique that researchers have explored is SLAM, or simultaneous localization and mapping. It refers to the ability of a robot or autonomous system to build a map of its environment while simultaneously determining its own location within that map. SLAM technologies have widespread adoption in various applications, ranging from self-driving cars, augmented and virtual reality, to drones and service robots. Advances in hardware and algorithms have enabled SLAM systems to become faster, more robust, and capable of operating in complex and dynamic environments. This thesis proposes a semantic SLAM approach that takes advantage of the system's ability to recognize objects and construct a map.

1.1 Significance and motivation

This thesis presents a novel system that integrates technologies like ROS middleware, YOLO object detection model, and the Gazebo simulation environment to solve the robot's localization problem based on landmarks localization and mapping. A study of different approaches is presented, including the semantic SLAM. After conducting the analysis, design, and implementation, the system provides a comprehensive solution that combines the strengths of each component to achieve accurate localization results.

1.2 Related works

Previous studies related to the presented thesis include an autonomous driving simulator by W. Cao *et al.* [4], a 3D object detection system for robotic arm grasping by C. -W. Chen *et al.* [5], and an automated detection system for a UAV by C. M'Sila *et al.* [6]. It also draws upon landmark-based localization and triangulation algorithms from various sources. A landmark-based localization system in an urban environment by X. Qu, B. Soheilian and N. Paparoditis [7], a triangulation toolbox based on open-source by S. Choi [8], a localization system for indoor mobile robots and AGVs by Loevsky and I. Shimshoni [2], a robot localization system based on visual landmarks by H. M. Ebied and M. S. Abdel-Wahab [9], a vision-based localization algorithm based on landmark matching, triangulation, reconstruction, and comparison by D. C. K. Yuen and B. A. MacDonald [10], a generalized geometric triangulation algorithm for a mobile robot by J. S. Esteves, A. Carvalho, and C. Couto [11].

1.2.1 ROS, YOLO and Gazebo for autonomous and robotic systems

W. Cao *et al*. [4] describe the development of an autonomous driving simulator built using the ROS framework with the Gazebo simulation environment and the YOLO object detection model. The simulator is designed to evaluate the performance of autonomous driving algorithms in a realistic simulation environment. The authors demonstrate that the simulator can accurately mimic real-world scenarios and can be used to train autonomous driving models effectively. ROS and Gazebo allow the integration of multiple software components and the ability to simulate a range of driving scenarios, while YOLO provides fast and accurate object detection. The results of this study highlight the potential for using this type of simulator for the development and testing of autonomous driving systems.

C. -W. Chen *et al.* [5] describe a system that uses ROS to control a robotic arm in a simulated environment created with Gazebo. The system uses the YOLO algorithm for 3D object detection to identify and locate objects within the simulated environment. The inverse kinematics algorithm then processes the detected objects to determine the necessary movements of the robotic arm to grasp the object. The system can successfully detect and grasp objects in the simulated environment, demonstrating the potential of this approach for real-world applications.

C. M'Sila *et al*. [6] discuss developing a system for detecting foreign object debris (FOD) using a UAV. The system uses ROS as the central software platform for controlling the UAV. In this case, ROS controls the UAV's flight and navigation. YOLO was used to detect FOD in the UAV's camera feed and then provide the necessary information for the UAV to navigate and inspect the area autonomously. Gazebo was used to simulate the UAV's environment and test the foreign object detection capabilities of the system. Overall, [1.2.3] discusses using ROS, YOLO, and Gazebo to create an automated foreign object debris detection system based on UAVs. These technologies are used together to provide a robust and efficient solution for detecting and inspecting FOD in a simulated environment.

1.2.2 Visual landmark-based systems

X. Qu, B. Soheilian, and N. Paparoditis [7] present a novel approach for object localization in urban areas using visual landmarks. The authors argue that traditional GNSS/GPS and inertial navigation systems are not always reliable in urban environments and propose a new system that uses a camera and a deep neural network to recognize landmarks in real-time. The network is trained on images of urban landmarks, and the GNSS coordinates dataset. The authors evaluate the system on a dataset of urban images. They show that it outperforms traditional GNSS and INS in terms of localization accuracy, is robust to changes in lighting conditions, and works well even in areas with limited GNSS coverage.

S. Choi [8] presents an open-source toolbox for benchmarking and evaluating landmark-based localization algorithms. The author provides a comprehensive collection of algorithms and datasets for landmark-based localization, including popular methods such as Extended Kalman Filters, Unscented Kalman Filters, and Particle Filters. The toolbox also includes evaluation metrics and scripts for comparing the performance of different algorithms. The author shows that the toolbox can be used for both research and practical applications, making it a valuable resource for researchers and practitioners in the field of landmark-based localization.

Loevsky and I. Shimshoni [2] present a new method for reliable and efficient landmark-based localization of mobile robots. The authors propose a system that combines multiple sensors, including cameras and laser rangefinders, to detect and recognize environmental landmarks. The system also employs a particle filter algorithm to estimate the robot's position and orientation based on the detected landmarks. The authors evaluate the approach on real-world datasets and show that it outperforms existing methods in terms of accuracy and computational efficiency. The results demonstrate the potential of the proposed method for reliable and efficient landmark-based localization of mobile robots in real-world environments.

1.2.3 Geometric landmark-based systems

H. M. Ebied and M. S. Abdel-Wahab [9] present a method for robot localization using visual landmarks. The authors propose a system that combines a camera and a triangulation algorithm to detect and recognize landmarks in the environment. The system uses the detected landmarks to estimate the robot's position and orientation, and the authors evaluate the the system's performance on real-world datasets. The results show that the proposed method is effective for robot localization based on visual landmarks and has the potential for use in various applications, such as autonomous navigation and mobile robotics.

D. C. K. Yuen and B. A. MacDonald [10] discuss the use of a landmark system to determine the position and orientation of a mobile robot. The system works by identifying and tracking landmarks in the environment and using them as reference points for estimating the robot's pose. The robot can detect landmarks in real-time. Once the landmarks are identified, the system employs a probabilistic framework to estimate the robot's pose based on geometric and visual cues obtained from the robot's sensors. The geometric cues are used to establish the relative positions of the landmarks, while the visual cues are used to verify and refine the estimate. This combination of geometric and visual cues allows the system to provide robust performance in the presence of occlusions and sensor noise.

J. S. Esteves *et. al* [11] present a new algorithm for mobile robot self-localization. The authors propose a generalized geometric triangulation method that can estimate the robot's position and orientation based on the observations of multiple landmarks in the environment. The algorithm is designed to handle arbitrary landmark configurations and can operate in real time. The authors evaluate the algorithm on both simulation and real-world datasets and show that it outperforms existing methods in terms of accuracy and computational efficiency. The results demonstrate the potential of the proposed algorithm for mobile robot self-localization in real-world environments.

1.3 Contribution

The present thesis uses image-guided landmark-based localization and mapping to determine a robot's position. The study presents the development, implementation, and evaluation of an approach to resolve the problem of robot positioning. The primary contributions of this thesis are:

1. Development of an architecture for a landmark-based localization and mapping system based on the SLAM approach.
2. Integration of ROS middleware and YOLOv5 object detection model with the Gazebo simulation environment.
3. Design and implementation of the components that support the mapping and computation of the robot’s position.
4. Design of the workflow that allows experimentation on the system.

1.4 Structure

- **Chapter 2** describes the theoretical foundations and key concepts that .guided the study, as well the development and implementation of the research.
- **Chapter 3** offers a detailed exposition of the different components used in the development of the system, as well as an explanation of the design decisions made during the development phase.
- **Chapter 4** focuses on a detailed description of the various components constituting the system. Additionally, it provides a comprehensive overview of the project's functionality.
- **Chapter 5** is about the experimental results of the project.
- **Chapter 6** brings the project’s conclusion and ideas for future research.

2 Background

The objective of this chapter is to provide a comprehensive overview of the foundational concepts, techniques, and technologies that form the basis of this work. By exploring the required background information, the reader will gain insight into the topic and the reasoning behind the research. This chapter aims to create a clear and defined context for the following discussions and analyses in the thesis.

2.1 SLAM

SLAM, or Simultaneous Localization and Mapping, is a fundamental concept in robotics and computer vision. It refers to the ability of a device to construct a map of its environment and determine its location within that map simultaneously. This is achieved by collecting sensory data from the device's sensors, such as cameras or lasers, and processing that data with algorithms to create a representation of the environment. As the device moves, the SLAM algorithm updates the map and the device's location within the map, allowing it to maintain an accurate understanding of its environment even as it moves. SLAM has been the subject of extensive research [12], with many algorithms and approaches proposed to address the various challenges. This research is driven by the need for accurate and efficient mapping and localization in applications such as autonomous vehicles, drones, and augmented reality.

2.1.1 Modern Visual SLAM

In their recent work, Xia *et al* introduce a modern V-SLAM system, as depicted in Figure 2.1. Such a system is commonly made up of the following components:

- Data acquisition through sensors, specifically image or video acquisition via cameras.
- Visual odometry, which estimates the robot's position and landmark locations based on successive frames in an image sequence.
- State estimation that globally estimates the state by fusing the results of visual odometry.
- Relocalization that enables the system to relocate when tracking fails or the map is reloaded.
- Loop closure detection, which maps the environment based on the task requirements.
- Mapping, which involves creating maps to meet the task requirements.

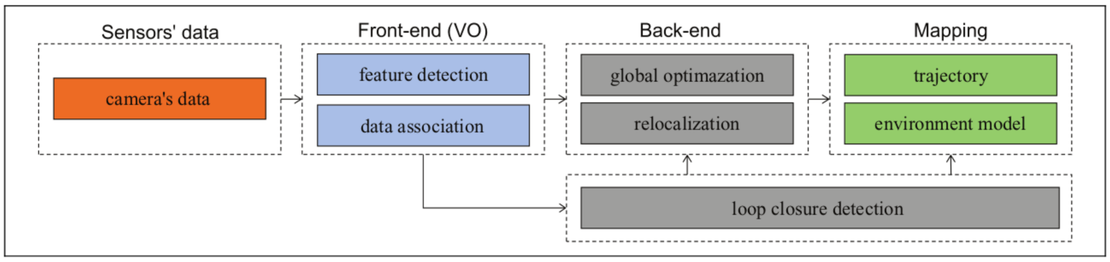


Figure 2.1: Representation of the architecture of a V-SLAM system. [12]

2.1.2 Hybridized SLAM

C. Debeunne and D. Vivet proposed a Hybridized SLAM model framework that consists of three main steps, as illustrated in Figure 2.2. The framework can be broken down into the following steps:

1) Data processing step.

- Performs feature detection and tracking for LiDAR and Camera.

2) Estimation step:

- Estimates the vehicle displacement from the tracked features using ICP, epipolar geometry, proprioceptive sensors, or a fusion of each, such as a Kalman filter or a multi-criteria optimization.

- Tries to detect and match landmarks from the map to the features.
- Refines the pose through filtering/optimization once matching is done.
- Estimates new landmarks.

3) Global Mapping Step:

- Determines whether the current data defines a key frame and brings in enough new information.
- Optimizes the trajectory locally or globally based on the detection of a loop closing.

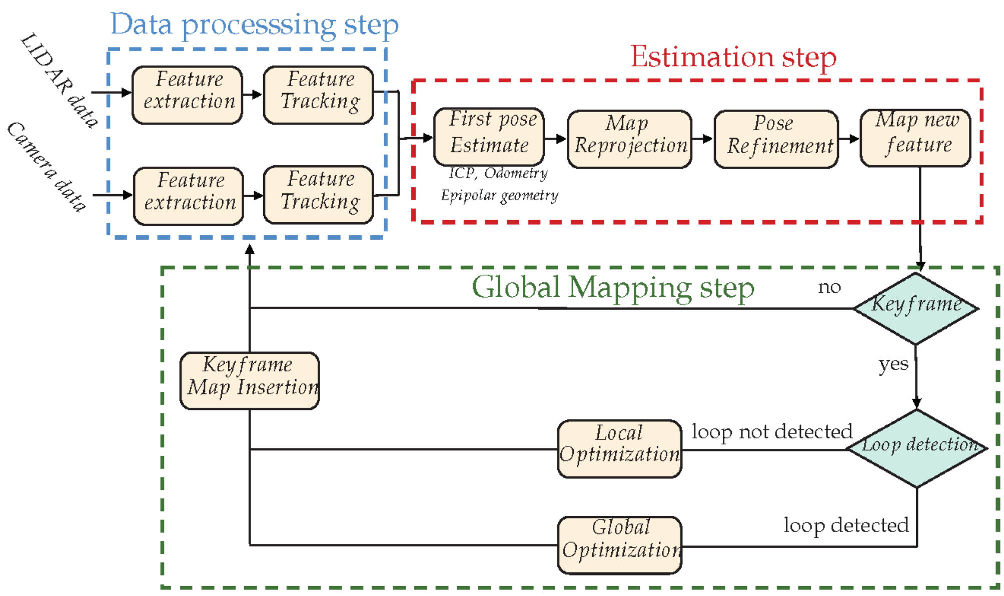


Figure 2.2: Hybridized SLAM global framework. [1]

2.1.3 Visual-based semantic SLAM

Visual-based semantic SLAM is a type of SLAM that combines visual data with semantic information to create a map of an environment and determine the device's location within that map. In visual-based semantic SLAM, the device's camera is used to capture images of the environment, and algorithms are used to process the images to extract both visual and semantic information. Visual information is used to create a map of the environment, while semantic information is used to label objects in the environment, such as furniture, doors, and walls. The combination of visual and semantic information enables the device to create a more detailed and accurate environment map, particularly useful in applications such as augmented reality and robotic navigation. [12]

Figure 2.3 depicts the architecture of a semantic SLAM framework, which establishes independent tracking for individual objects within a 3D scene. This design allows for efficient feature selection and data association, from 2D to 3D and from single thread to multi-thread, leading to improved VO robustness and accuracy in practice.

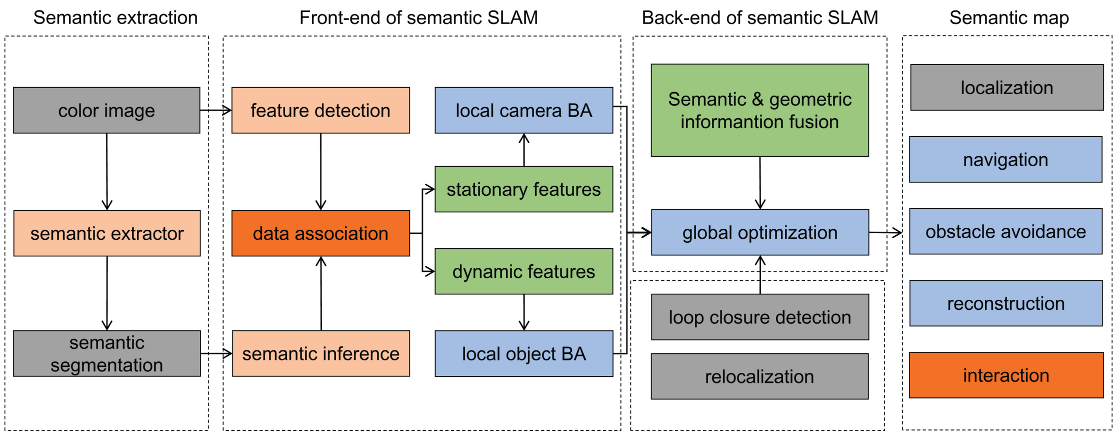


Figure 2.3: Architecture of a semantic SLAM system. [12]

2.2 Sensor fusion

Sensor fusion is the process of combining information from multiple sensors to create a more complete and accurate representation of the environment. In the context of camera and LiDAR sensors, sensor fusion combines the strengths of both sensors to provide a more robust and accurate perception of the surroundings.

LiDAR sensors measure distances to objects by emitting laser beams and measuring the time it takes for the reflections to return. They provide a dense and precise 3D representation of the environment, which is helpful for obstacle detection, mapping, and localization tasks.

Camera sensors, on the other hand, capture images of the environment, providing information about objects' color, texture, and shape. They are widely used in computer vision tasks such as object recognition, tracking, and segmentation.

By combining the information from both sensors, the limitations of each can be overcome. For example, lighting conditions can affect cameras and are less accurate than LiDAR for measuring distances. At the same time, LiDAR can struggle with objects that have low reflectivity or are transparent. By fusing the information from both sensors, a more robust and accurate perception of the environment can be obtained.

Several techniques for fusing camera and LiDAR data include point cloud registration, projection of LiDAR points onto the image, image segmentation and LiDAR point classification, and deep learning approaches.

2.2.1 Sensor fusion techniques for camera and LiDAR

Some of the techniques used for camera and LiDAR sensor fusion include point cloud registration, projection of LiDAR points onto image, image segmentation and LiDAR point classification, and deep learning approaches [13,14]. Each of the previous techniques has its advantages and limitations, and the choice of the best method depends on the application's specific requirements. For example, point cloud registration is a good choice for applications requiring precise alignment between sensors. At the same time, deep learning approaches may be more suitable for applications where complex relationships between the sensors need to be learned.

2.2.1.1 Point cloud registration

A process of aligning the point cloud data from the LiDAR sensor with the image data from the camera. This process can be done by finding correspondences between features in the point cloud and the image and using them to estimate the transformation between the two sensors.

2.2.1.2 Projection of LiDAR points onto image

The technique involves projecting the 3D points from the LiDAR point cloud onto the camera's image plane. This technique allows for integrating the spatial information from the LiDAR and the color and texture information from the camera.

2.2.1.3 Image segmentation and LiDAR point classification

This technique involves segmenting the image into different regions and classifying the points in the LiDAR point cloud based on their location relative to the segments. This information can then be used to improve the accuracy of object detection and segmentation algorithms.

2.2.1.4 Deep learning

Recent advances in deep learning have enabled the development of end-to-end models that can fuse camera and LiDAR data and perform tasks such as object detection, semantic segmentation, and depth estimation. These models can learn to exploit the complementary information from both sensors to achieve improved performance.

2.3 Robot Operating System

The acronym ROS stands for Robot Operating System, despite its categorization as a middleware platform for developing robotic applications rather than an operating system. This open-source platform is both multi-domain and multi-platform. In the context of this project, the version of ROS2 Foxy was utilized. This updated version provides a comparable interface to ROS1, built on top of DDS, a software middleware that facilitates a publish-subscribe transport system.

2.3.1 ROS nodes

A ROS node can be described as an individual execution process similar to a standalone program. Multiple ROS nodes operate simultaneously and interact through a primarily event-driven mechanism, utilizing shared data.

2.3.2 ROS topics

The fundamental structure of ROS is based on the publish/subscribe architecture, as depicted in Figure 2.4. This concept is similar to that found in software packages such as MQTT, ZeroMQ, RabbitMQ, or ModBus. However, unlike these other packages, ROS does not utilize a broker. Instead, in ROS1, there is a ROS master that facilitates node discovery, while in ROS2, nodes can discover each other through automated discovery mechanisms as it operates in a distributed fashion.

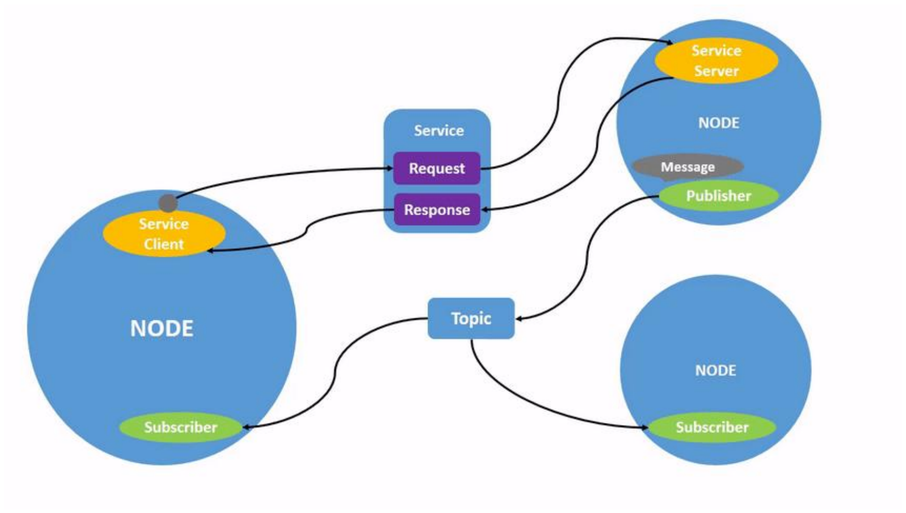


Figure 2.4: Communication framework used by ROS. [15]

2.3.3 ROS Messages

ROS provides pre-defined messages that standardize the representation of common data types, such as sensor data, navigation data, and geometric data. Standardizing these data types enables effortless communication between nodes programmed in different languages, as they can be efficiently serialized. Furthermore, the ability to record these messages to files, known as "bags" or "rosbags", offers a valuable resource for debugging and studying the performance of a ROS system over time. The standardization of data and ease of communication between nodes written in different programming languages make ROS an indispensable tool for developing of advanced robotic systems.

2.3.4 SDF and URDF files

The Simulation Description Format (SDF) is a file format used to describe the structure, kinematics, dynamics, and sensors of robots for simulation and visualization [16]. SDF is natively supported by the Gazebo simulator, providing an API for loading and manipulating SDF models. The SDF format uses a custom XML-based format, making it simple to parse and manipulate data. The previous makes it an efficient and effective solution for describing robots in simulation environments, providing a straightforward way to define the behavior and appearance of robotic systems for a variety of applications.

It is important to note that SDF is not the only file format used for robot description. The Unified Robot Description Format (URDF) is another commonly used format developed by the ROS community [17]. While URDF and SDF both serve similar purposes, there are differences between the two formats in terms of complexity and structure. URDF is generally considered a more complex format, while SDF is simpler and better suited for use with the Gazebo simulator. Ultimately, the choice between URDF and SDF will depend on the specific requirements of the simulation task.

2.4 YOLOv5

YOLO, an acronym for “You Only Look Once” is an object detection model. The characteristics of YOLOv5 are lightweight, easy to use, quick training and inference, good performance, and versatility, making the model suitable for real-time object detection. YOLOv5 offers four models, namely s, m, l, and x, each having different detection accuracy and performance, as shown in the graph from Figure 2.5

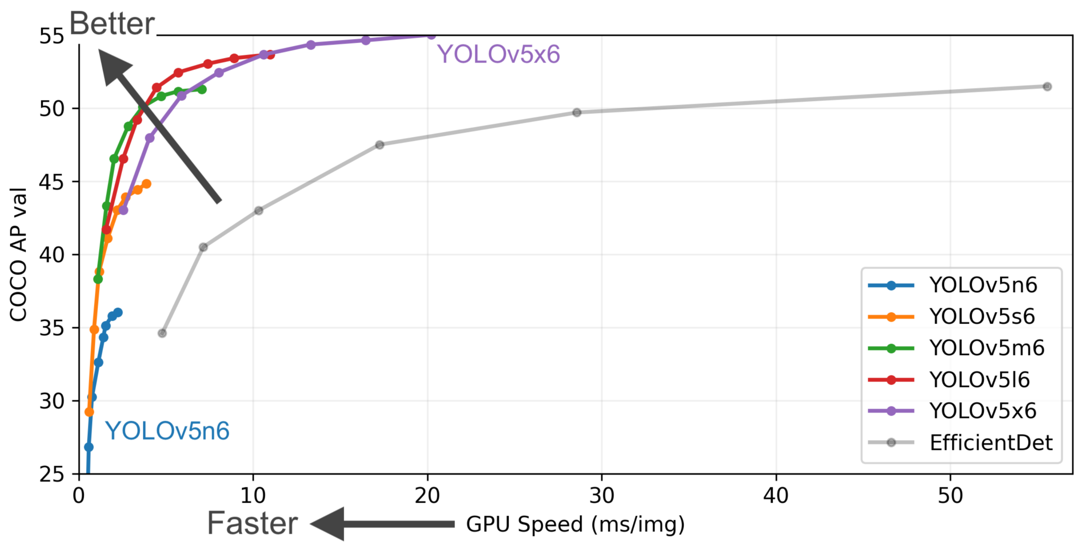


Figure 2.5: YOLOv5 versions comparison. [18]

2.4.1 YOLOv5 architecture

Figure 2.6 displays the architecture of YOLOv5, which comprises three distinct components: the Backbone, which is CSPDarknet; the Neck, which is PANet; and the Head, which is the YOLO layer.

An overview of the model’s versions is presented in Table 2.1. Inference speed on CPU, GPU , and the number of parameters for an image size of 640 pixels.

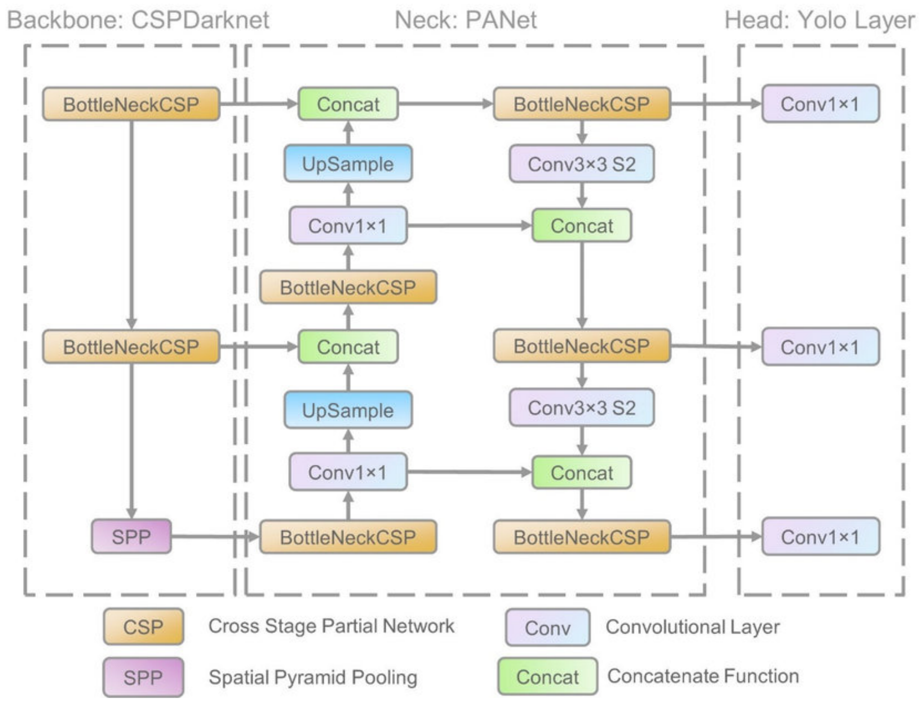


Figure 2.6: Architecture overview of YOLOv5. [19]

Model Name	Params (Million)	Accuracy (mAP 0.5)	CPU Time (ms)	GPU Time (ms)
YOLOv5n	1.9	45.7	45	6.3
YOLOv5s	7.2	56.8	98	6.4
YOLOv5m	21.2	64.1	224	8.2
YOLOv5l	46.5	67.3	430	10.1
YOLOv5x	86.7	68.9	766	12.1

Table 2.1: Pre-trained checkpoints. [18]

2.4.2 Datasets

Customized datasets can be utilized to train machine learning models. For this purpose, YOLOv5 and Ultralytics offer tools to facilitate dataset labeling. In the context of this project, the training phase was omitted.

2.4.2.1 COCO

The Common Objects in Context (COCO) dataset is a well-known resource widely used for training and evaluating object detection algorithms. The COCO dataset contains more than 328,000 images and labels across 80 object categories and has been used to train object detection models, including YOLOv5. [20]

2.4.2.2 COCO 128

In addition to the full COCO dataset, there is also a smaller version known as COCO 128 [21], which contains a subset of 128 categories from the original COCO dataset. COCO 128 provides a more manageable dataset for training and evaluating object detection models while providing a diverse set of object categories to work with.
The COCO 128 dataset has been used to evaluate the performance of object detection algorithms, including YOLOv5. It is an adequate dataset for training and testing object detection models. The COCO 128 dataset provides a good balance between dataset size and complexity, making it a valuable resource for the computer vision community.

2.5 Simulation environment

A simulation environment is a software tool that facilitates the creation of a virtual replica of a real-world system. This virtual representation can be utilized for testing, experimentation, and analysis without impacting the system.

2.5.1 Gazebo simulator

Gazebo is a 3D dynamic simulator that enables the simulation of robot’s behavior in complex indoor and outdoor environments in accurately and efficiently. Like game engines, Gazebo uses physics simulation with a higher degree of fidelity, a collection of sensors, and user interfaces. The common uses of Gazebo are: testing robotics algorithms, designing robots, and performing regression testing with realistic scenarios. Some key features of Gazebo are multiple physics engines, a rich library of robot models and environments, a wide variety of sensors, and convenient programmatic and graphical interfaces.

2.5.2 Turtlebot robot

A Gazebo TurtleBot simulation is a virtual representation of the TurtleBot robot created using the Gazebo simulation environment. In a Gazebo TurtleBot simulation, the TurtleBot is modeled in 3D, and its various components and sensors are simulated to behave as they would in the real world. This virtualization allows developers, researchers, and educators to test and validate their algorithms and control systems in a safe and controlled environment without needing a physical TurtleBot.

Using Gazebo to simulate a TurtleBot, users can perform various tasks such as mapping, navigation, and object recognition. They can even test multi-robot scenarios in which multiple TurtleBots interact with each other in a simulated environment. The previous can provide valuable insights and help users fine-tune their algorithms and control systems before deploying them on physical robots.

Figure 2.7 shows two versions of the Turtlebot robot, while Figure 2.8 portrays the simulated representation of the Turtlebot Waffle Pi.

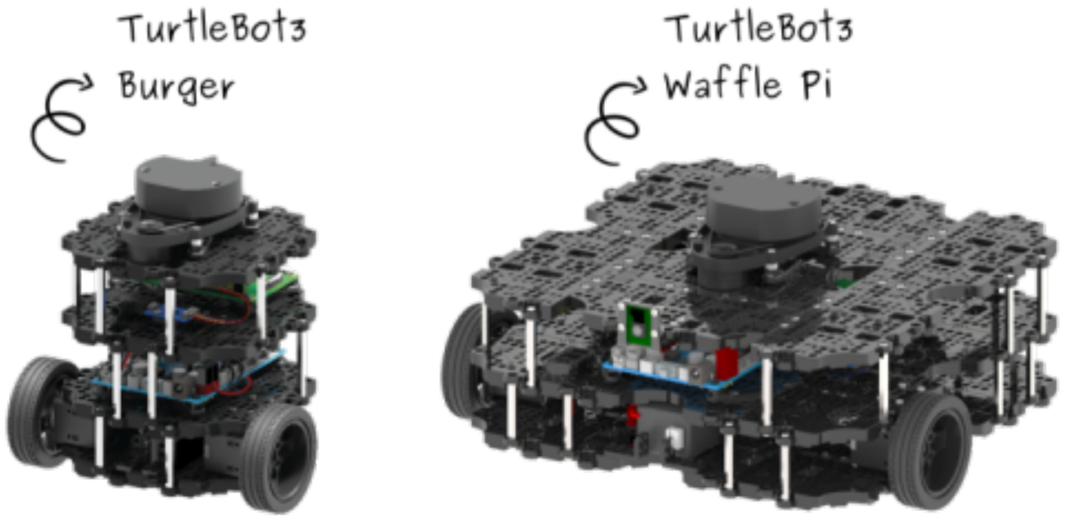
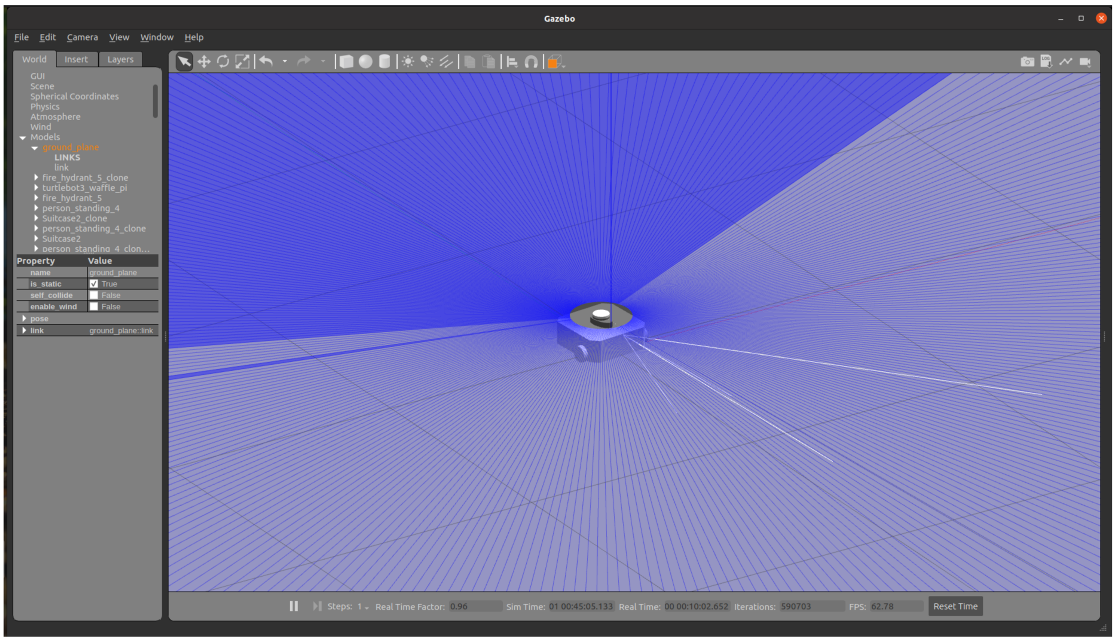


Figure 2.7: Turtlebot robots. Burger and Waffle Pi models. [22]



Property	Value
name	ground_plane
is_static	✓ True
self_collide	False
enable_wind	False
pose	
link	ground_plane::link

```
Steps: 1  Real Time Factor: 0.96  Sim Time: 01 00:45:05.133  Real Time: 00 00:10:02.652  Iterations: 590703  FPS: 62.78
```

Figure 2.8: Turtlebot Waffle Pi in Gazebo.

2.6 The Levenberg-Marquardt algorithm

The Levenberg-Marquardt algorithm is a well-known optimization method for solving nonlinear least-squares problems [23]. Due to its versatility, it has gained widespread use within the field of robotics. It has been employed in various problems related to SLAM, such as mapping, navigation, and applications for autonomous vehicles [24].

Within the context of SLAM, the algorithm is applied recurrently to revise the estimates of both the robot's pose and the positions of landmarks, where a probabilistic framework such as an extended Kalman filter or particle filter is utilized. In each iteration, the Levenberg-Marquardt algorithm is applied, resulting in an updated estimation of the Robot's pose and the positions of the landmarks.

3 Design overview

In this chapter, the software design decisions of the system are presented and discussed. First, the system and its phases are described in a high-level view. Then we describe the components that integrate the system, such as the simulator, movement control for the robot, object recognition, object localization & mapping, and finally, the robot's position computation.

3.1 System overview

The diagram in Figure 3.1 provides a simplified illustration of the system's structure, highlighting its key components. The system comprises four modules: Simulation, Control, Recognition, and Mapping/Localization. These modules are designed to work together to accomplish the desired outcome, each playing a unique role. The ROS middleware supports the functionality of the system.

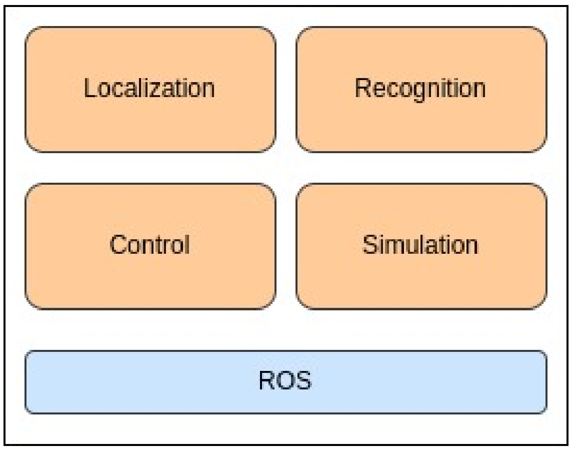


Figure 3.1: Functional view.

The system comprises two phases: Map Formation and Robot's Position Computation. Figure 3.2 illustrates the processes within each stage.

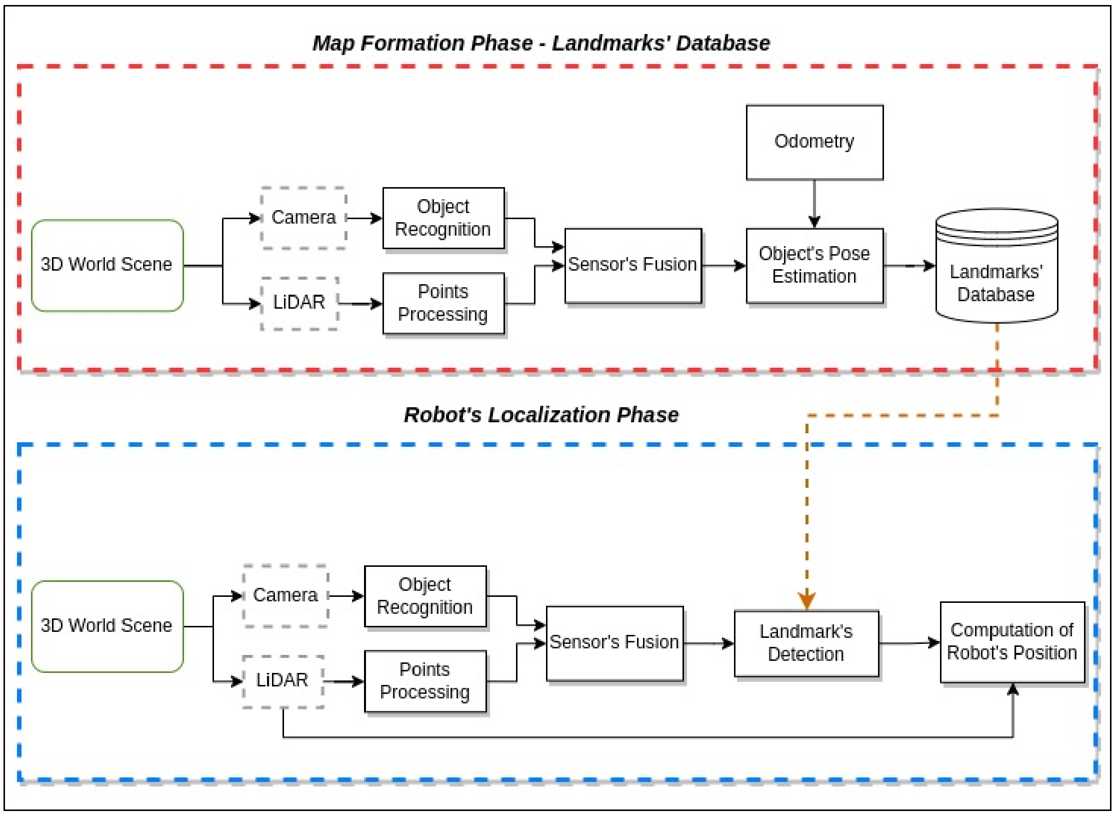


Figure 3.2: Map Formation Stage and Robot's Localization Stage.

3.1.1 Analysis of SLAM approaches

Section 2.1 from the previous chapter introduced the SLAM models. Making a comparison between those models, V-SLAM, Hybridized SLAM, Semantic SLAM, and our system, we find the following similarities:

3.1.1.1 Comparison versus V-SLAM

When comparing our approach to the V-SLAM model, we found that five characteristics match: data acquisition, visual odometry, state estimation, loop closure, and mapping. Our approach features an additional component, the LiDAR sensor. In our system, the relocalization component is triggered by the user.

3.1.1.2 Comparison versus Hybridized SLAM

After comparing our approach with the Hybridized SLAM model, we found that both models comprise the three main stages of data processing, evaluation, and global mapping—however, the main difference lies in the evaluation stage. While the Hybridized SLAM model requires the estimation of the vehicle displacement from the tracked features, our approach manages this step by matching the landmark's database with the landmark's detector, thus eliminating the need for vehicle displacement estimation.

3.1.1.3 Comparison versus Semantic SLAM

In our comparison of the Semantic SLAM model, we identified how our system manages most of the processes in each model's stages. Our approach handles the color image, semantic extractor, and semantic segmentation processes during the semantic extraction stage. Our approach manages feature detection, data association, and semantic inference in the front-end stage. In this stage, the optimization step for local camera BA and local object BA, as well as stationary/dynamic features, is simplified by our system through the use of points' processing operation. In the back-end stage, our approach covers semantic and geometric information fusion, loop closure detection, and relocalization triggered by the user. Finally, in the semantic map stage, our approach handles localization and reconstruction while interaction is possible but, in our case, user-triggered. It is important to note that our approach does not cover global optimization in the back-end stage or navigation and obstacle avoidance in the semantic map stage.

3.2 Simulator component

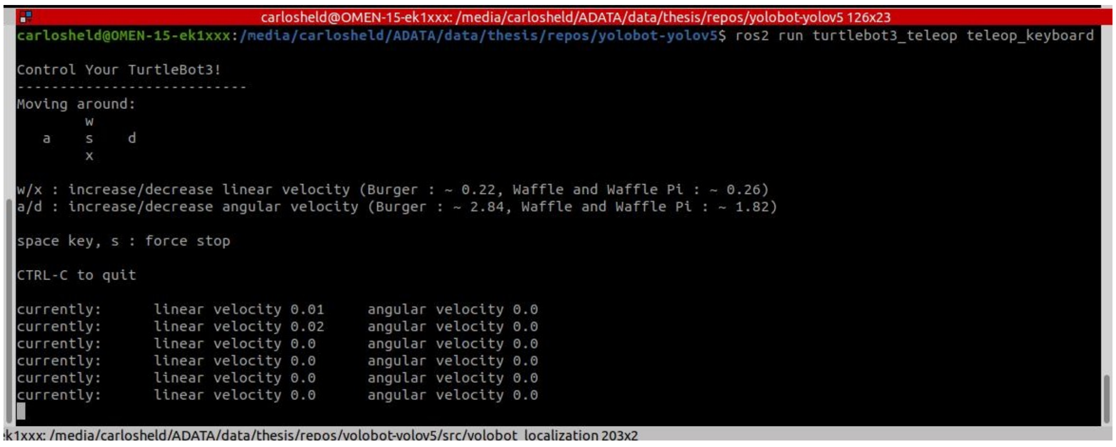
Gazebo is the simulator component's back-end and was selected for its compatibility with ROS and advantageous features. To integrate the system into a 3D simulation, fundamental elements, including the world environment, the robot, sensors, and landmark objects, must be included. However, for this project, the object recognition component, which depends on object geometry, restricted the selection of object models available in the environment. Each object underwent testing and selection based on the detection confidence level to ensure accurate detection. Examples of how the objects were tested are provided in *Appendix A***,** where it can be observed the utilization of the RViz tool during the development process.

3.2.1 World simulation configuration

For the virtual environment, a custom SDF world file was written. The elements included in the XML-like file are the Turtlebot-Waffle Pi robot, the sensors, and the landmark objects. The objects were imported manually in the simulation environment. Still, it is essential to consider that every time a new object is introduced, a piece of code configuration is introduced in the SDF file. The new object configuration needs to be saved for future use.

3.3 Control component

The Control component provides the motion control of the robot, operated by the user. Figure 3.3 shows the Teleop keyboard. The keyboard provides a simple interface via the Ubuntu terminal that allows the user to control the speed and direction of the robot's movement with five keys. This interface allows the user to conveniently increase or decrease the speed and direct the robot's movement.



```
carlosheld@OMEN-15-ek1xxx: /media/carlosheld/ADATA/data/thesis/repos/yolobot-yolov5 126x23
carlosheld@OMEN-15-ek1xxx:/media/carlosheld/ADATA/data/thesis/repos/yolobot-yolov5$ ros2 run turtlebot3_teleop teleop_keyboard

Control Your TurtleBot3!
---------------------------
Moving around:
        w
   a    s    d
        x

w/x : increase/decrease linear velocity (Burger : ~ 0.22, Waffle and Waffle Pi : ~ 0.26)
a/d : increase/decrease angular velocity (Burger : ~ 2.84, Waffle and Waffle Pi : ~ 1.82)

space key, s : force stop

CTRL-C to quit

currently:      linear velocity 0.01     angular velocity 0.0
currently:      linear velocity 0.02     angular velocity 0.0
currently:      linear velocity 0.0      angular velocity 0.0
currently:      linear velocity 0.0      angular velocity 0.0
currently:      linear velocity 0.0      angular velocity 0.0
currently:      linear velocity 0.0      angular velocity 0.0

k1xxx: /media/carlosheld/ADATA/data/thesis/repos/yolobot-yolov5/src/yolobot_localization 203x2
```

Figure 3.3: Teleop keyboard for operating the robot.

3.4 Recognition component

The objective of the Recognition component is to perform the object recognition task. To this end, the YOLOv5 model was integrated into the ROS 2 package. Leveraging the COCO 128 dataset [21], the model could successfully recognize up to 128 classes of objects. Table 2.1 presents the weights for inference. In our case, the selected weights correspond to the YOLOv5x model. According to the model parameters, the accuracy level is 68.9, slightly higher than the previous version. Processing time analysis reveals that the CPU takes 766 ms, while the GPU takes 12.1 ms. The YOLOv5x model was evaluated alongside other models in a simulation environment and met the requirements. The NVIDIA GTX 1650 GPU was used as the hardware in this project.

Appendix B lists the principal software packages and their versions used for the project development.

3.5 Localization and Mapping component

The Localization and Mapping component consists of the RGB camera and LiDAR as the primary data input sources, which provide the acquisition of data from the environment. Notably, both sensors are integrated into the Gazebo simulator and configured as part of the SDF file.

3.5.1 Robot’s localization

This section concentrates on the localization component of the robot, with Figure 3.4 illustrating the problem at hand. The localization problem can be divided into phases: Map Formation and Robot Localization. During the Map Formation phase, the robot, denoted by the blue circle labeled "Robot pose," traverses the environment and records observations of objects landmarks represented by orange triangles. In the Robot Localization phase, denoted by the gray circle labeled "New robot pose," the robot changes its position by an amount Δx.

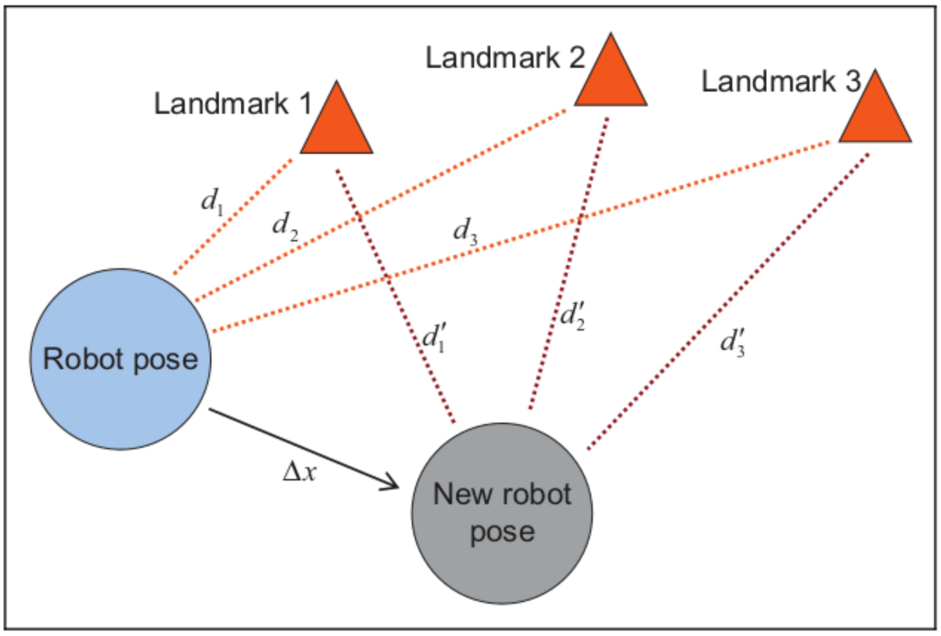


Figure 3.4: Visual representation of the localization problem. [12]

The fundamental issue in the second phase is to determine the robot's new position, utilizing the information gathered during the Map Formation phase.

3.5.1.1 Triangulation

The problem of how to solve the robot's position based on visual landmarks is an approach that has been used and is well-defined. A practical example of how this approach was used is presented by H. M. Ebied and M. S. Abdel-Wahab [9]. The triangulation algorithm was studied by M. Betke and L. Gurvits [25]. In their work, they consider an autonomous agent that uses a map to navigate through an environment containing landmarks. Those landmarks are marked on the agent's maps. The autonomous agent has a tool for measuring angles, in the case of our system, the LiDAR sensor. As stated by M. Betke and L. Gurvits, the agent may use the following algorithm to identify its location in the environment:

1) identify surrounding landmarks in the environment;
2) find the corresponding landmarks on the map;
3) measure the bearings of the landmarks relative to each other;
4) compute your position efficiently.

In Figure 3.5, a graphical illustration of the triangulation problem is presented. The solution to this problem requires measuring the distances between the robot and each object and the knowledge of the position of each landmark on the recalled map.

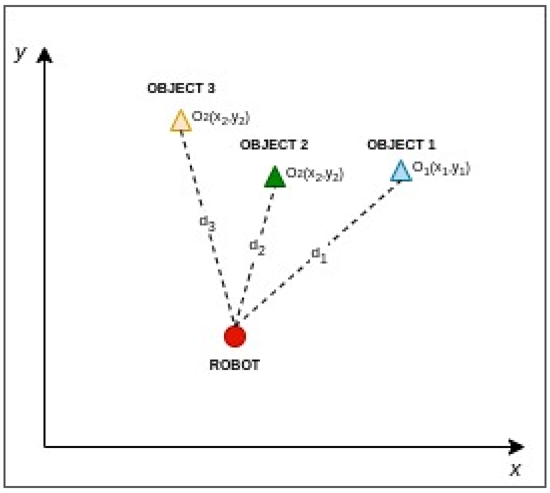


Figure 3.5: Triangulation between the robot and landmarks.

As per H. M. Ebied nd M. S. Abdel-Wahab's work, we have implemented Equations (4), (5), and (6) in our research. These equations pertain to calculating the distance between the robot's position and the landmarks.

$$d_1^2 = (x_1 - x_{robot})^2 + (y_1 - y_{robot})^2 \tag{3.1}$$

$$d_2^2 = (x_2 - x_{robot})^2 + (y_2 - y_{robot})^2 \tag{3.2}$$

$$d_3^2 = (x_3 - x_{robot})^2 + (y_3 - y_{robot})^2 \tag{3.3}$$

The system determines the distances between the robot and each of the three landmarks (d_n,where: n=1,2,3). These distances are known variables given that the data from the environment is gathered by the sensor readings and processed by the system. Additionally, the location of each landmark is known, as it was recorded during the map formation phase. As such, the pairs of coordinates $Object_n(x_n, y_n)$ where, n = 1,2,3; corresponding to each landmark, are also established.

3.5.1.2 Solution of the triangulation system of equations

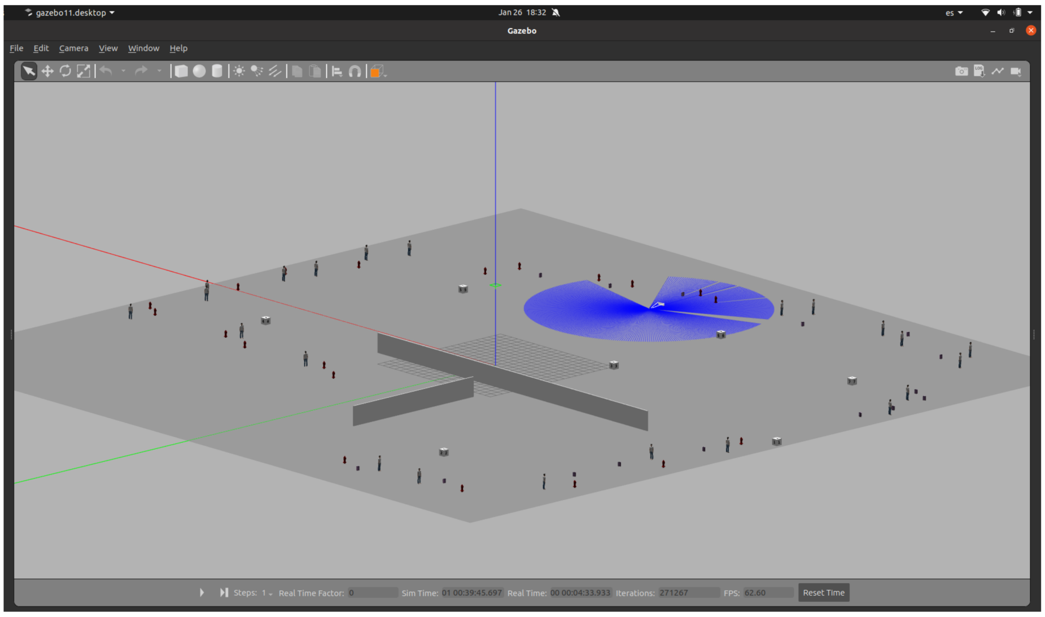
As asserted by M. Betke and L. Gurvits, the system of nonlinear equations (3.1, 3.2, and 3.3) can be solved using a least squares method. The computational cost is O(n) or linear because it depends on the number of landmarks to be processed [9].

Considering the nonlinearity of the equations system, we found that the Levenberg-Marquardt algorithm represents a plausible solution supported by prior studies [24,26].

4 Implementation and experimental setups

This chapter provides an in-depth description of the system implementation and delineates the various functional elements that constitute the system. It presents a comprehensive overview of the experimental setups utilized in the implementation process and details the procedures followed to ensure their successful execution.4.1 Simulation setup

The simulation setup is based on the Gazebo software. As previously detailed in section 3.2.1, the robot, RGB camera, and LiDAR were defined in a ROS package. The objects were selected and placed (section 3.2) within the simulation environment, as depicted in Figure 4.1. It was essential to distribute the objects such that no object visually obstructed one another from the robot's viewpoint. The simulation comprised 20 groups, each containing three objects.



Steps: 1 Real Time Factor: 0 Sim Time: 01 00:39:45.697 Real Time: 00 00:04:33.933 Iterations: 271267 FPS: 62.60

Figure 4.1: Bird's eye view of the Gazebo simulation with the robot and objects.

4.2 Objects’ recognition with the YOLOv5 model

Algorithm 1 outlines the steps involved in the object recognition process. The RGB camera provides the input data for the process. Subsequently, the YOLOv5 model is utilized tor recognize objects and generate a detection list. The detection list is published from the objects' recognition as a ROS topic. The detection list is transmitted as a ROS topic, containing the indispensable information to represent the attributes of each object, such as the class, the confidence level of the YOLO detection, and the data from the bounding box description. The bounding box is an array of four elements: [top left x-coordinate, top left y-coordinate, bottom right x-coordinate, bottom right y-coordinate]. The Sensor's Fusion module then receives this list for further processing.

ALGORITHM 1: OBJECTS' RECOGNITION

```
Input: image - RGB camera image (640x480)
Output: objects_detection_list -> [[object_1_class, confidence_level, coordinates_of_bounding_box], [object_2_class, confidence_level, coordinates_of_bounding_box], [object_3_class, confidence_level, coordinates_of_bounding_box]]
ConfigureCameraSubscriber(weights="yolov5x.pt", inference_size=640, confidence_threshold=0.25)
InitializeDevice("NVIDIA GTX 1650")
LoadModel(weights="yolov5x.pt")
while camera_callback(image) do
    prediction_1 = Inference(image)
    prediction_2 = ApplyNMS(prediction_1)
    prediction_3 = ApplyClassifier(prediction_2)
    objects_detection_list = ProcessDetections(prediction_3)
end
publish_topic(objects_detection_list)
```

Figure 4.2 shows an inference run for four distinct object classes: fire hydrant, person, stop signal, and suitcase. The assigned confidence levels for each object were 0.91, 0.93, 0.93, and 0.81, respectively.

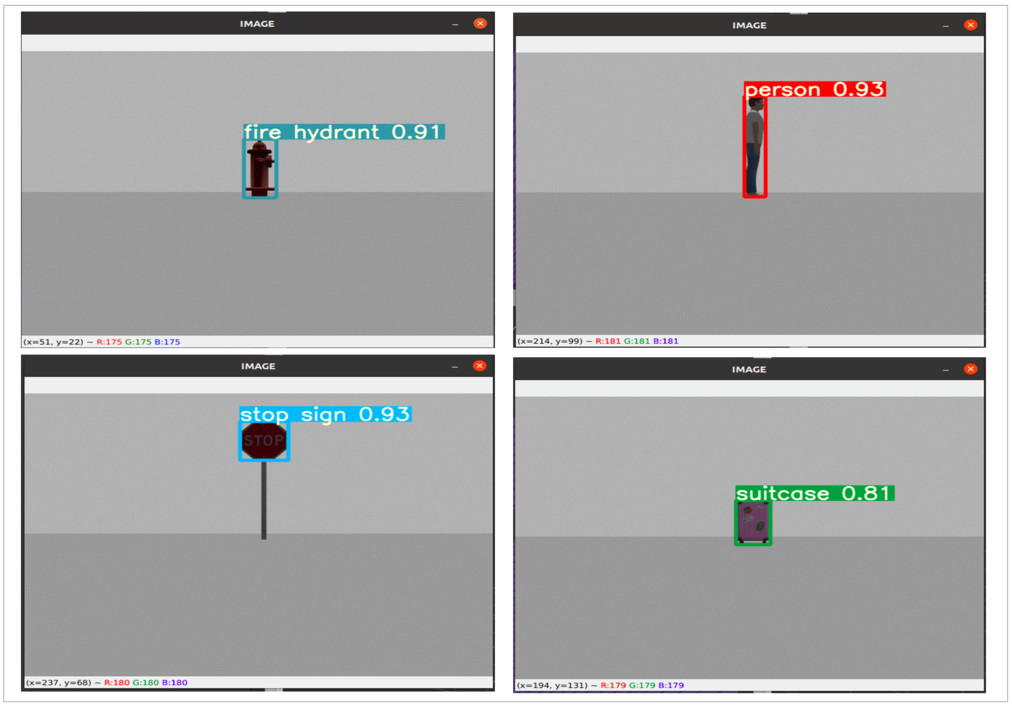


IMAGE
fire hydrant 0.91
person 0.93
stop sign 0.93
suitcase 0.81
(x=51, y=22) ~ R:175 G:175 B:175
(x=214, y=99) ~ R:181 G:181 B:181
(x=237, y=68) ~ R:180 G:180 B:180
(x=194, y=131) ~ R:179 G:179 B:179

Figure 4.2: Objects detected by the YOLOv5 model in the Gazebo simulation.

4.3 LiDAR points processing

Algorithm 2 has the objective of processing the LiDAR points. This process receives data from the LiDAR sensor and generates a binary version of the LiDAR data alongside the list of angles where objects are detected. Subsequently, the outputs of this process are connected to the Sensor's Fusion process. It is essential to note that, initially, the LiDAR's field of view must be filtered to correspond with the RGB camera's FoV. The selected FoV ranges are limited to 0-30 and 330-360 degrees. Concerning the distance, detection occurs within the range of 0.5 to 15 meters.

ALGORITHM 2: POINTS' PROCESSING

```
   Input: LiDAR scan topic
   Output: lidar_binary, objects_pose_lidar
1  lidar_ranges = ExtractRanges(LIDAR scan topic)
2  for lidar_ranges do
3      filtered_view_range = FilterViewRange(lidar_ranges)
4  end
5  publish_topic(filtered_view_range)
```

```
if filter_view_range then
    ranges_filter_binary = FormRangesFilterBinary(filtered_view_range)
end

for n=list(ranges_filter_binary) do
    lidar_binary = CheckPattern(ranges_filter_binary)
    if list[i]==1 then
        ranges_object_index = ranges_object_index + i
    end
end

for list(ranges_object_index) do
    index = list[i]
    objects_pose_lidar = [index, filtered_view_range[index]]
end
```

Figure 4.3 illustrates the data ranges the system can detect from RGB camera and LiDAR perspectives. Although the hardware specifications [27] of the Laser Distance Sensor (LDS-01) used by the Turtlebot specify a distance range of 120 mm to 3500 mm, the simulation environment permits the use of the illustrated ranges through configuration.

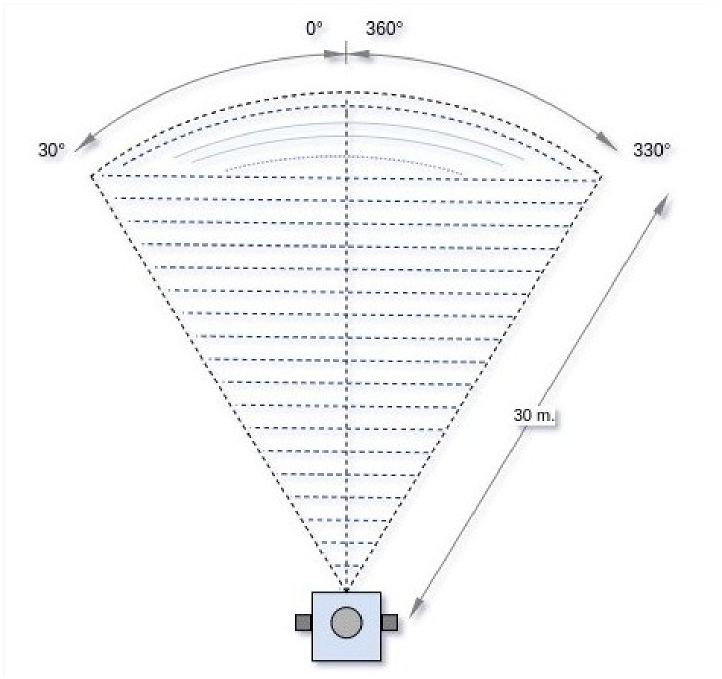
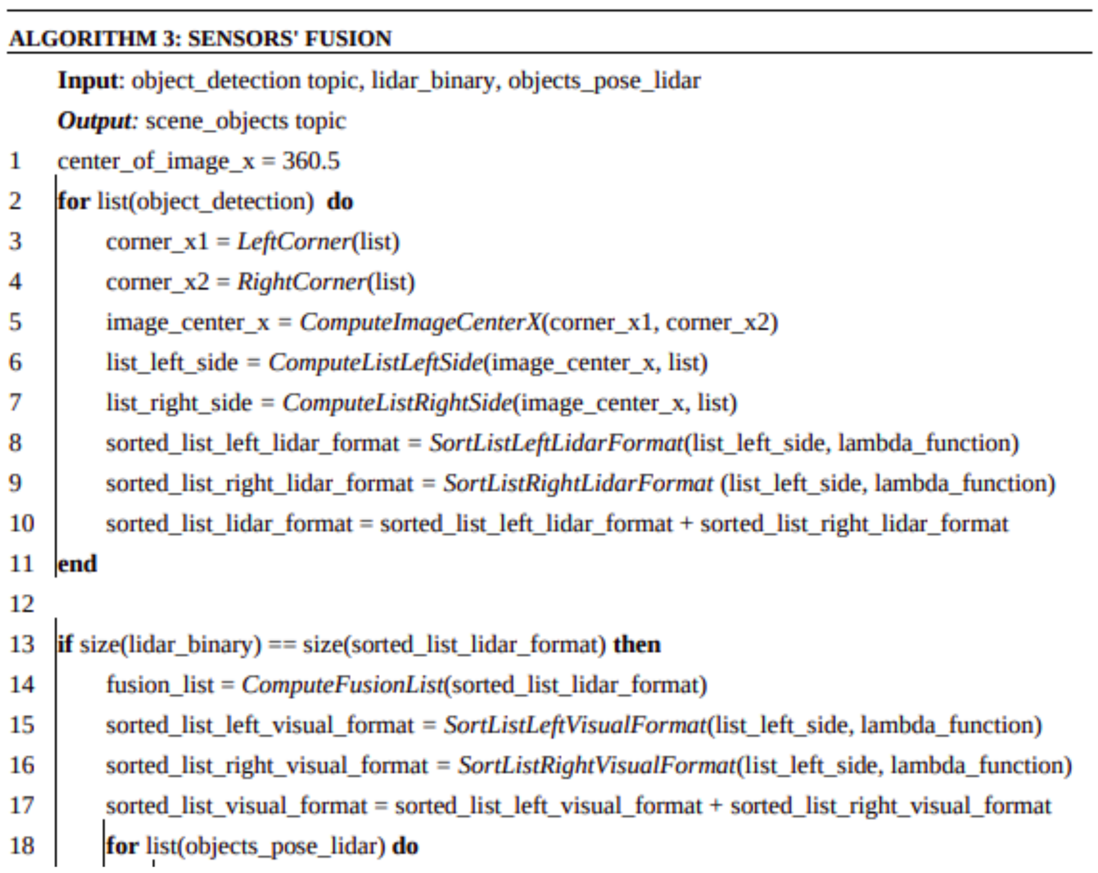


Figure 4.3: LiDAR ranges from the top view. Distance and FoV.

4.4 Sensors' fusion

This system process detailed, by Algorithm 3, can be regarded as the back-end of the semantic SLAM approach, which fuses semantic and geometric information. Algorithm 3 is used to unify the input data from the RGB camera, which already includes semantic information and the processed data from the LiDAR sensor. Since the data from both sensors are presented in distinct formats, it is crucial to establish a consistent data format. To illustrate the previous, Figure 4.4 depicts a scene from the robot's perspective. The data from both sensors should be represented consistently from left to right, with the fire hydrant being the first object to appear, followed by the person and the suitcase. The output of this process contains a list of objects and their positioning relative to the robot. The subsequent process - Objects' Position Estimation, receives and processes this list.



ALGORITHM 3: SENSORS' FUSION

```
Input: object_detection topic, lidar_binary, objects_pose_lidar
Output: scene_objects topic
center_of_image_x = 360.5
for list(object_detection) do
    corner_x1 = LeftCorner(list)
    corner_x2 = RightCorner(list)
    image_center_x = ComputeImageCenterX(corner_x1, corner_x2)
    list_left_side = ComputeListLeftSide(image_center_x, list)
    list_right_side = ComputeListRightSide(image_center_x, list)
    sorted_list_left_lidar_format = SortListLeftLidarFormat(list_left_side, lambda_function)
    sorted_list_right_lidar_format = SortListRightLidarFormat (list_left_side, lambda_function)
    sorted_list_lidar_format = sorted_list_left_lidar_format + sorted_list_right_lidar_format
end

if size(lidar_binary) == size(sorted_list_lidar_format) then
    fusion_list = ComputeFusionList(sorted_list_lidar_format)
    sorted_list_left_visual_format = SortListLeftVisualFormat(list_left_side, lambda_function)
    sorted_list_right_visual_format = SortListRightVisualFormat(list_left_side, lambda_function)
    sorted_list_visual_format = sorted_list_left_visual_format + sorted_list_right_visual_format
    for list(objects_pose_lidar) do
```

```
        if 0 <= list <= 30 then
            objects_lidar_data_left = objects_lidar_data_left + list
        end

        if 330 <= list <= 360 then
            objects_lidar_data_right = objects_lidar_data_right + list
        end
    end
    objects_pose_visual = SortListVisualFormat(objects_lidar_left, objects_lidar_right)
    object_1 = FormObject(sorted_list_visual_format, objects_pose_visual, index=0)
    object_2 = FormObject(sorted_list_visual_format, objects_pose_visual, index=1)
    object_3 = FormObject(sorted_list_visual_format, objects_pose_visual, index=2)
    scene_list = object_1 + object_2 + object_3
end
publish_topic(scene_list)
```

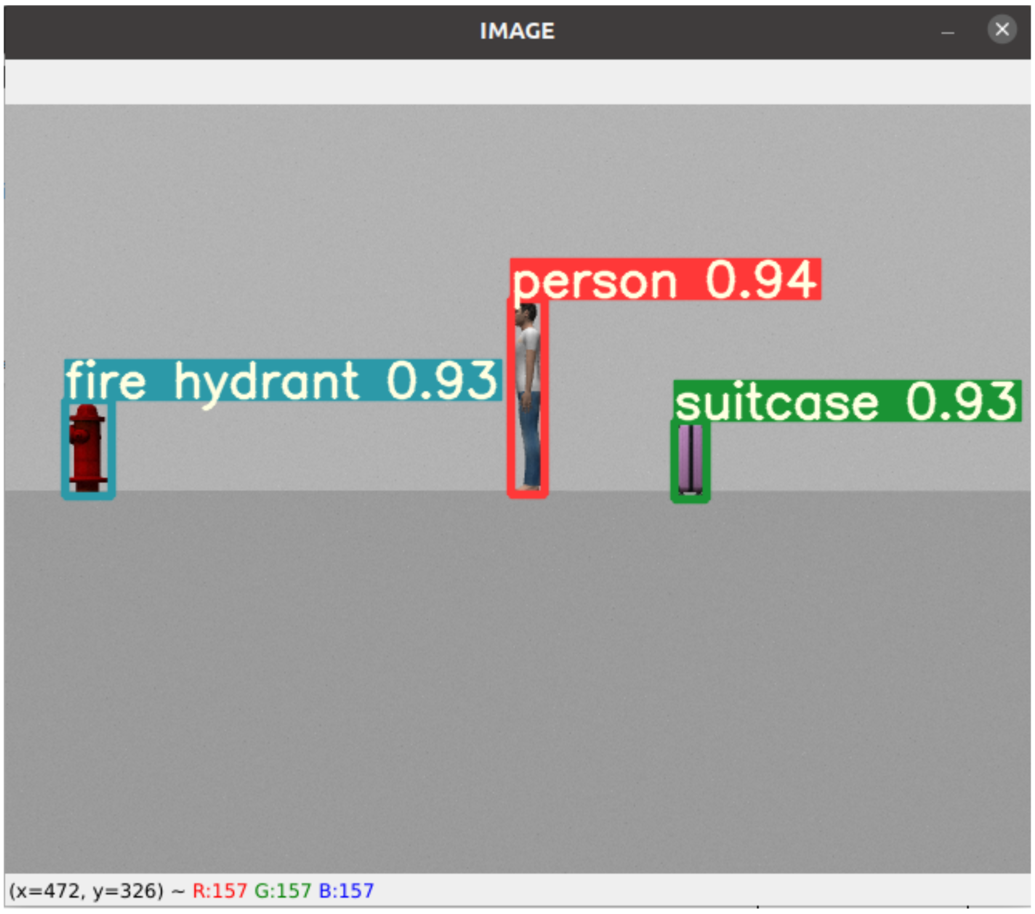
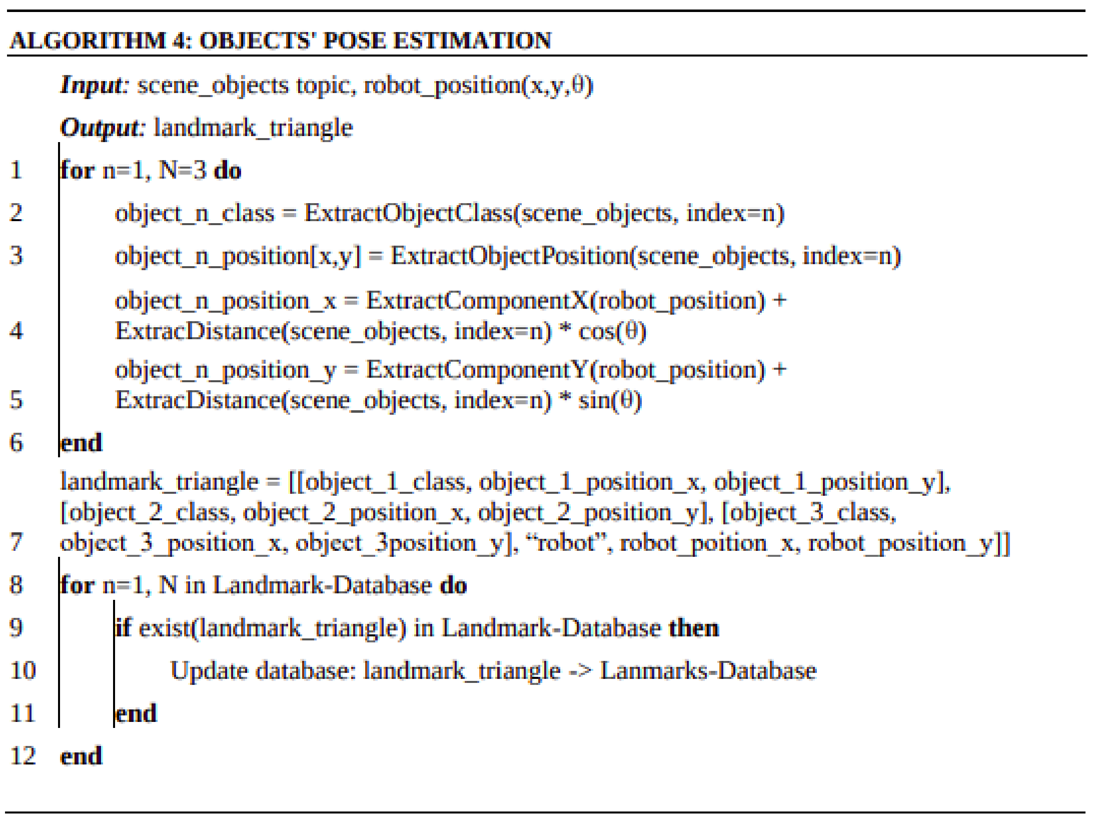


Figure 4.4: Robot’s view of an object’s scene.

4.5 Objects' position estimation

This process detailed in Algorithm 4 is present in the Map Formation phase. It receives inputs from the Sensors' Fusion block and the odometer, utilizing this information to estimate the position of an object in relation to the robot's location. The output of this process is a set of landmark objects, in other words, a group of three objects with their corresponding localization information. The set of landmark objects is updated in the Landmarks Database each time a new group of elements is discovered. A group is regarded as new if at least one object class differs from that in other groups or if the order of the objects is distinct from that of other groups.



ALGORITHM 4: OBJECTS' POSE ESTIMATION

```
    Input: scene_objects topic, robot_position(x,y,θ)
    Output: landmark_triangle
1   for n=1, N=3 do
2       object_n_class = ExtractObjectClass(scene_objects, index=n)
3       object_n_position[x,y] = ExtractObjectPosition(scene_objects, index=n)
4       object_n_position_x = ExtractComponentX(robot_position) +
        ExtracDistance(scene_objects, index=n) * cos(θ)
5       object_n_position_y = ExtractComponentY(robot_position) +
        ExtracDistance(scene_objects, index=n) * sin(θ)
6   end
7   landmark_triangle = [[object_1_class, object_1_position_x, object_1_position_y],
    [object_2_class, object_2_position_x, object_2_position_y], [object_3_class,
    object_3_position_x, object_3position_y], "robot", robot_poition_x, robot_position_y]]
8   for n=1, N in Landmark-Database do
9       if exist(landmark_triangle) in Landmark-Database then
10          Update database: landmark_triangle -> Lanmarks-Database
11      end
12  end
```

Appendix C presents the resulting Landmarks Database (Map).

4.6 Landmarks’ detection

The process elaborated in Algorithm 5 is present in the Robot's Localization phase. It replicates the Objects' Position estimation process from the Map Formation phase. Rather than using the odometer information, it uses the Landmarks Database generated in the earlier phase to search for correspondences within groups of landmarks. If a match is found, the system will trigger the execution of the subsequent process, which is the computation of the robot's position.

ALGORITHM 5: LANDMARKS' DETECTION

```
   Input: scene_objects topic, robot_position(x,y,θ) , landmark_triangle
   Output: matched_landmark_element, find_match_flag
1  for n=1, N=3 do
2      object_n_class = ExtractObjectClass(scene_objects, index=n)
3      object_n_position_x = ExtractComponentX(robot_position) +
       ExtracDistance(scene_objects, index=n) * cos(θ)
4      object_n_position_y = ExtractComponentY(robot_position) +
       ExtracDistance(scene_objects, index=n) * sin(θ)
5  end
6  landmark_triangle = [[object_1_class, object_1_position_x, object_1_position_y],
   [object_2_class, object_2_position_x, object_2_position_y], [object_3_class,
   object_3_position_x, object_3position_y], “robot”, robot_poition_x, robot_position_y]]
7  for n=1, N in Landmark-Database do
8      matched_landmark_element = FindMatch(landmark_triangle)
9  end
10 Update register: matched_landmark_element -> Landmark_Register
11 if exist(matched_landmark_element) then
12     publish_topic(find_match_flag = "1")
13 end
```

4.7 Computation of robot’s position

Algorithm 6, the final process in the second phase, requires the activation of a flag from the preceding step. Upon detecting the flag, the process accesses the register containing the matched group of landmarks and gets the LiDAR data from the scene within the robot's field of view. After, the algorithm employs the Levenberg-Marquardt algorithm, as detailed in section 3.5.1.2, to solve the system of nonlinear equations and obtain a solution for the coordinates of the robot's position $P_{robot}(x_{robot},y_{robot})$ with respect to the map.

ALGORITHM 6: COMPUTATION OF ROBOT'S POSITION

```
    Input: find_match_flag, Landmark-Register-File, Visual-Detection-Register-File
    Output: solution$[x, y]$
1   if find_match_flag == 1 then
2       Initial guess: $x_0 = [k_1, k_2]$
3       solution$[x, y]$ = Root(function_solver, $x_0$, method = "Levenberg-Marquardt")
4   end
5
6   function_solver($x_{robot}, y_{robot}$)
7       Open(Landmark-Register-File)
8       $d_1$ = object_position_visual_right
9       $d_2$ = object_position_visual_center
10      $d_3$ = object_position_visual_center
11      Open(Visual-Detection-Register-File)
12      $x_1$ = object_position_landmark_right_x
13      $y_1$ = object_position_landmark_right_y
14      $x_2$ = object_position_landmark_center_x
15      $y_2$ = object_position_landmark_center_y
16      $x_3$ = object_position_landmark_left_x
17      $y_3$ = object_position_landmark_left_y

        Equations =
18      $[(x_1 - x_{robot})^2 + (y_1 - y_{robot})^2 - d_1^2]$,
        $[(x_2 - x_{robot})^2 + (y_2 - y_{robot})^2 - d_2^2]$,
        $[(x_3 - x_{robot})^2 + (y_3 - y_{robot})^2 - d_3^2]$.
19  return Equations
```

Appendix D presents the obtained data of the computation of the robot's position.

5 Results

This chapter presents the outcomes of the robot's position computation and its subsequent analysis. The experimental results obtained are discussed, providing a comprehensive understanding of the findings.

The first step involved obtaining groups of landmarks from the environment to form a map, as presented in *Appendix C* with the Landmarks Database. A depiction of the arrangement of objects in the Gazebo world can be observed in Figure 5.1 from a top-down view, and a similar scene from a bird's eye view is in Figure 4.1.

Figure 5.2 displays the Cartesian plot of the robot's trajectory in the Gazebo environment and the detected objects during the mapping phase. The graph comprises

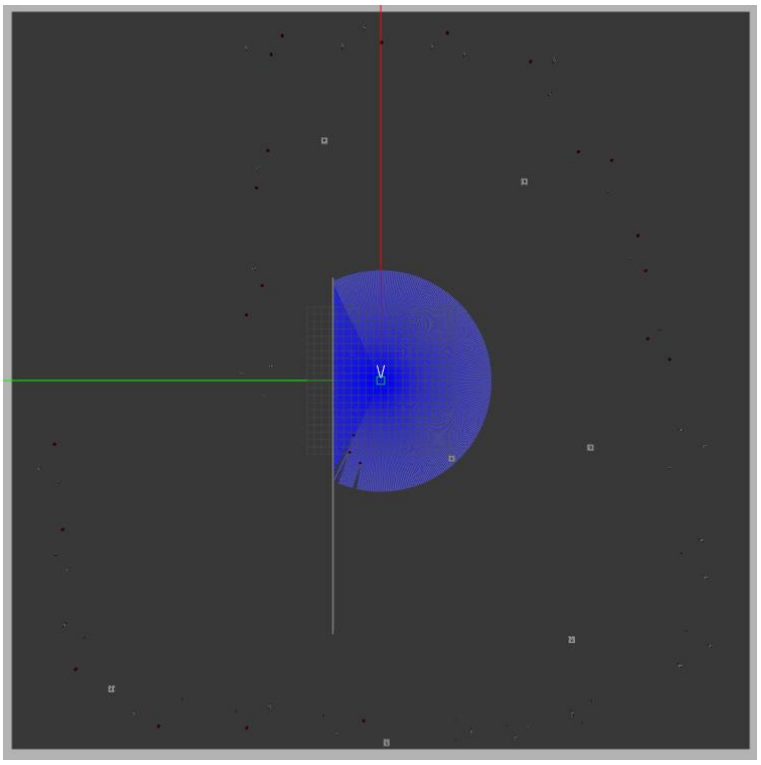


Figure 5.1: Top view of the Gazebo simulation.

twenty groups of objects, each composed of three blue crosses that signify the location of the detected objects. The robot's path is represented by black lines, while a dot mark indicates the position where the scene was captured. During the mapping phase, sixty objects were accurately identified for each of the twenty distinct groups. Further details regarding the formation of these groups can be found in *Appendix B*.

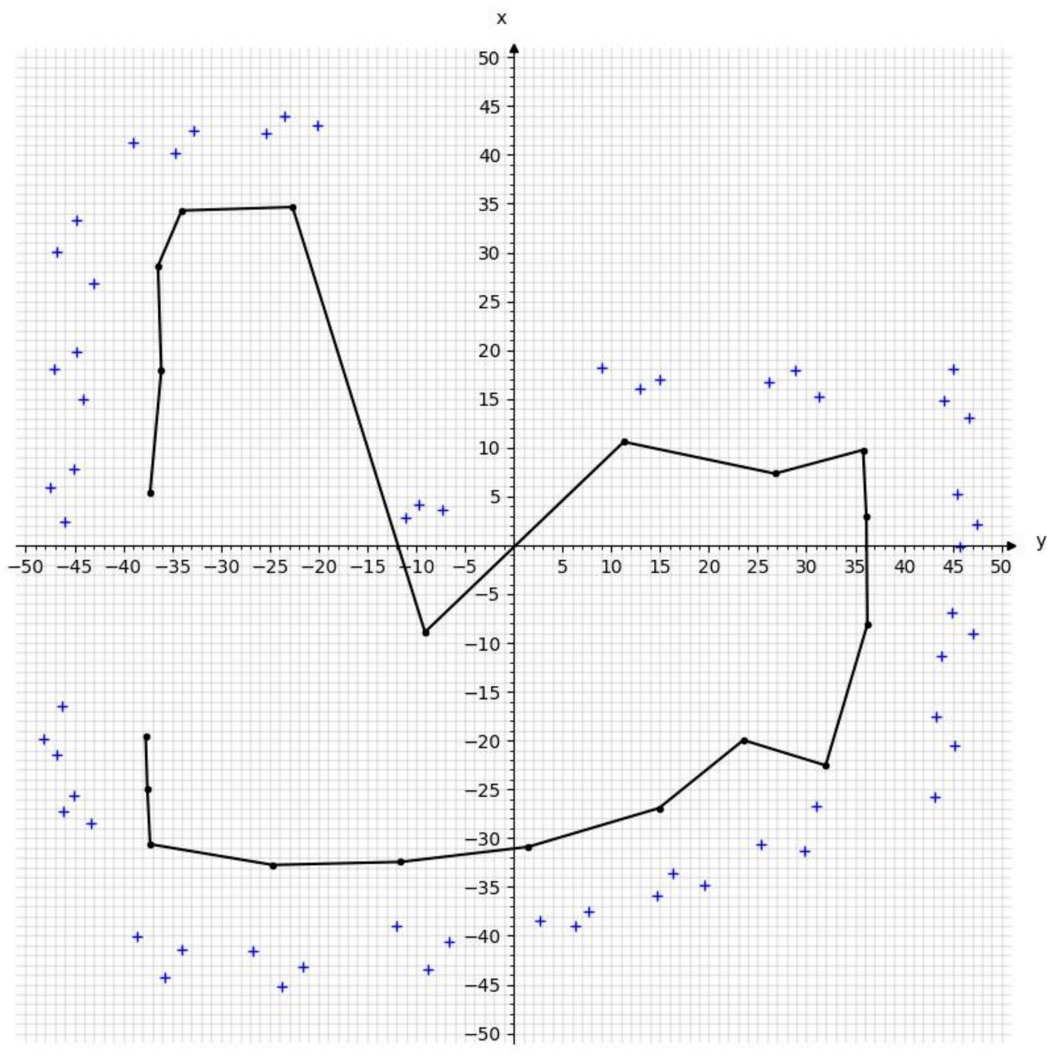


Figure 5.2 Trajectory of the robot in the Gazebo simulation – top view

The second step was to direct the robot to different locations to obtain a view of the environment and compute the robot’s position.

5.1 Quantitative Analysis

Following the computation of the robot’s position, a quantitative analysis of the results was performed. The detail of the collected data for the experimental results can be found in *Appendix D*.

The success of the computation of the robot's position was determined by verifying that the error between the position of the robot's ground truth provided by the odometric information and the position estimated by the system was less than 5% of the relative error in each axis, namely x, and y.

Figure 5.3 provides a visual representation of the data, with blue and red crosses denoting the robot’s position according to the ground truth and computed position, respectively. As indicated, the blue and red crosses are in close proximity, implying that the computed distance is close to the ground truth.

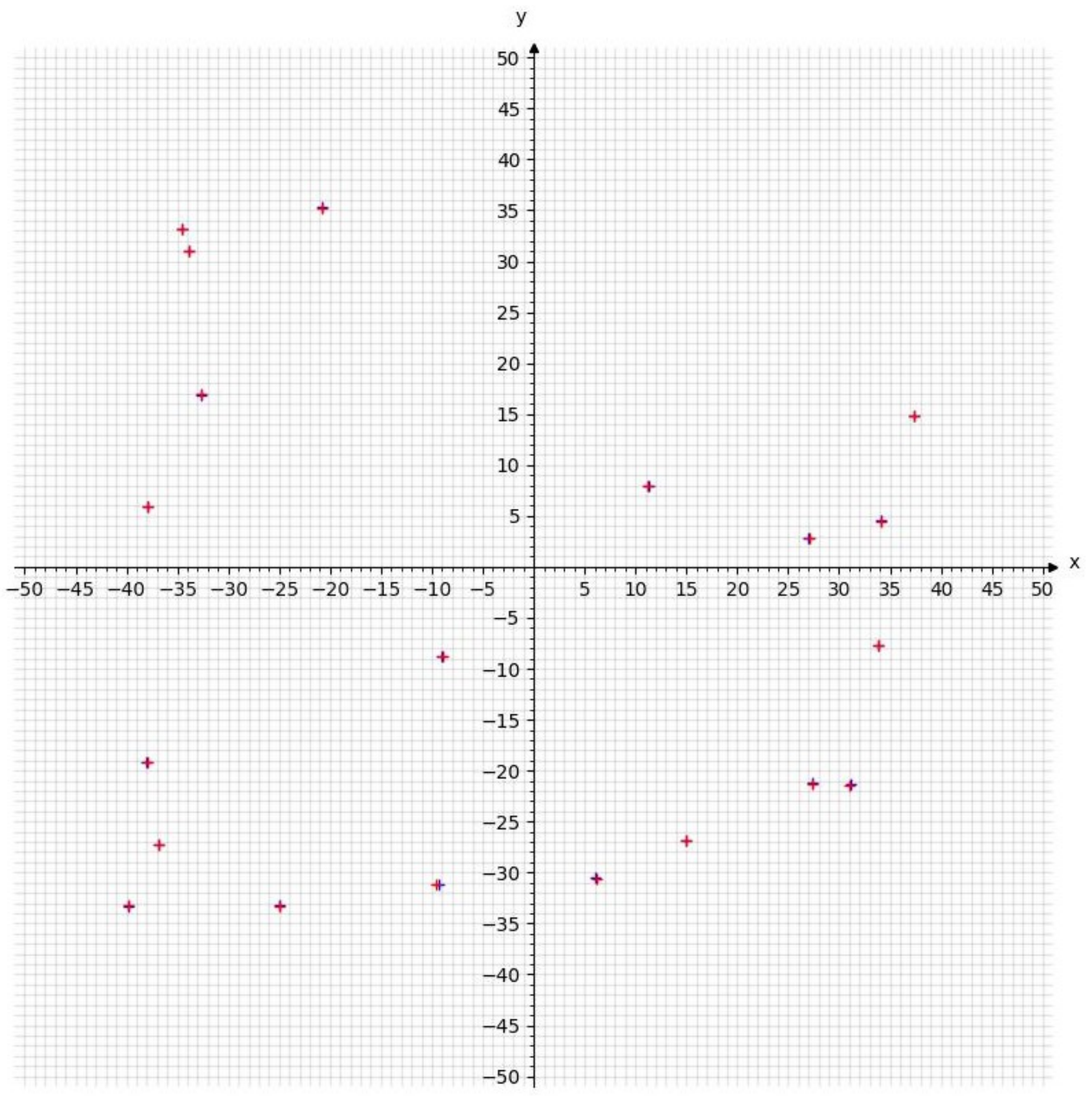


Figure 5.3: Robot's position – ground truth (blue) vs. computed (red)

In terms of localization error, the calculated mean error was found to be 0.34% on the x-axis and 0.49% on the y-axis. In order to visualize the distribution of localization errors, boxplots were created for both the x-axis and y-axis, presented in Figures 5.4 and 5.5, respectively. Analysis of the boxplot for the x-axis revealed a median absolute error of 0.01 m (1 cm.). In comparison, the median absolute error for the y-axis was calculated to be 0.015 m (1.5 cm.).

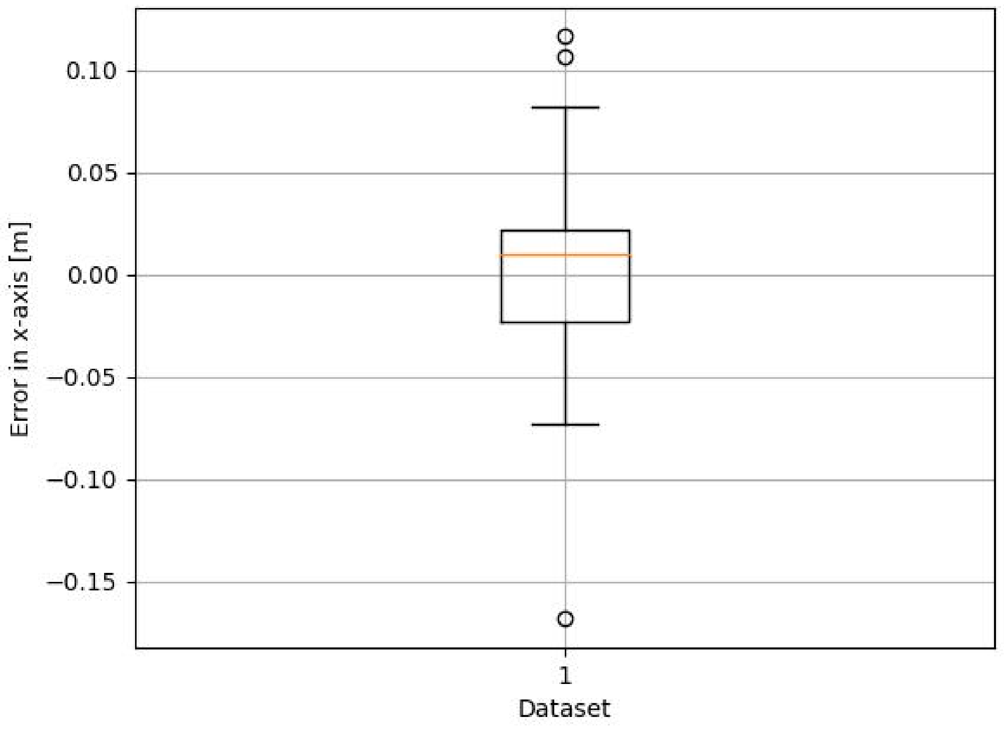


Figure 5.4: Boxplot of the localization error in the x-axis.

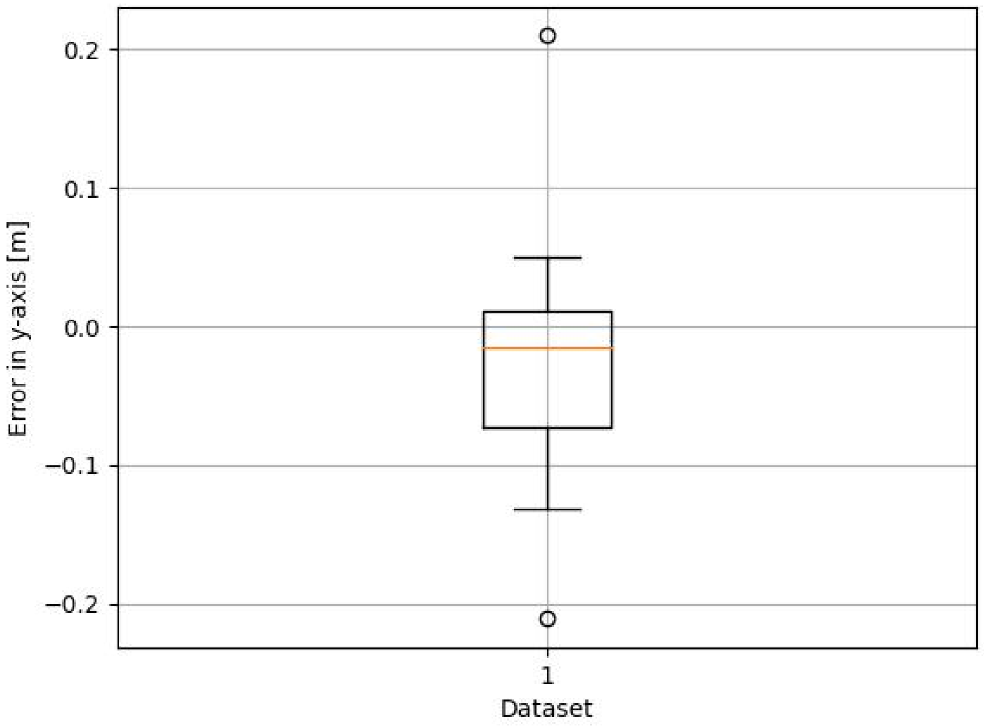


Figure 5.5: Boxplot of the localization error in the y-axis.

As part of our analysis of results, we compared our outcomes with those of previous studies [25, 9] that utilized the triangulation algorithm for robot position estimation. Specifically, we reviewed two studies that used this method, including M. Betke and L. Gurvits' analysis of 200 robot positions which revealed mean errors of 0.93 cm and 10.6 cm for the x-axis and y-axis, respectively. In contrast, H. M. Ebied and M. S. Abdel-Wahab reported a mean error of 0.35 cm for the euclidean distance for the same algorithm based on a dataset of 143 robot positions. Notably, both studies incorporated noisy data in their experiments.

5.2 Qualitative Analysis

This section presents a qualitative analysis of the data and observations obtained during the experimentation to identify the sources of inaccuracies in the system. The analysis indicates that the primary cause of these inaccuracies was the challenges encountered in extracting measurements from the environment using the sensors.

During the camera testing, it was observed that the object recognition model misclassified some objects due to the camera's perception of their shape. The previous could cause faults for the fusion algorithms, leading to errors in map formation and, consequently, in the robot's position estimation. However, it is worth mentioning that only objects correctly classified by the recognition component were included in the experiment.

Regarding the LiDAR sensor, we observed that the system's perception of an object is influenced by two main factors: the object's geometric characteristics and the LiDAR sensor's detection mechanism. The object's shape and thickness between others fall under the first category, while the robot's position affects the number of points the LiDAR sensor can detect in the second category. It should be noted that LiDAR virtualization used in the simulation is a simplified representation of a physical LiDAR, resulting in a lower density of points provided by the sensor. The lower density of points can impact the accuracy of the robot's position estimation and should be considered when implementing the system.

6 Conclusion

In conclusion, the study presented in this thesis has successfully demonstrated the viability of image-guided landmark-based localization as a method for robot localization. By exploring various approaches, including SLAM, and designing and implementing a comprehensive system that integrates components such as localization, control, recognition, and simulation, the effectiveness of the proposed method was verified through experimentation in a simulated environment. The study's results confirmed that image-guided landmark-based localization is a viable and effective means of solving the problem of robot localization.

6.1 Future research

In order to build upon the work presented in this thesis, future research should consider several paths for improvement and expansion. These paths include implementing the system in a real-world environment, which would provide further insight into the feasibility and effectiveness of image-guided landmark-based localization in a more complex and dynamic setting.

In addition, future research could focus on training the model to recognize a broader range of representative objects, allowing the system to process different environments and scenarios better. The previous could also involve improving the robustness of the sensor fusion algorithm for LiDAR, enabling more effective recognition of objects in the environment.

Furthermore, studies may explore ways to enhance the system's ability to make better choices based on the geometry of objects detected or segmented. The enhancement could lead to improved decision-making in complex and cluttered environments.

Finally, increasing the size of the dataset and conducting quantitative benchmarking of the system would provide a more comprehensive evaluation of its performance and enable comparison with existing methods. This insight could identify areas for improvement and guide further system development.

7 Appendix

7.1 Appendix A

Test of a bicycle in RViz tool. On the left side of Figure 7.1, the view from the Gazebo simulator is displayed. The image in the center shows the labeled object from the YOLO detection, and the right screen shows the points detected in RViz.

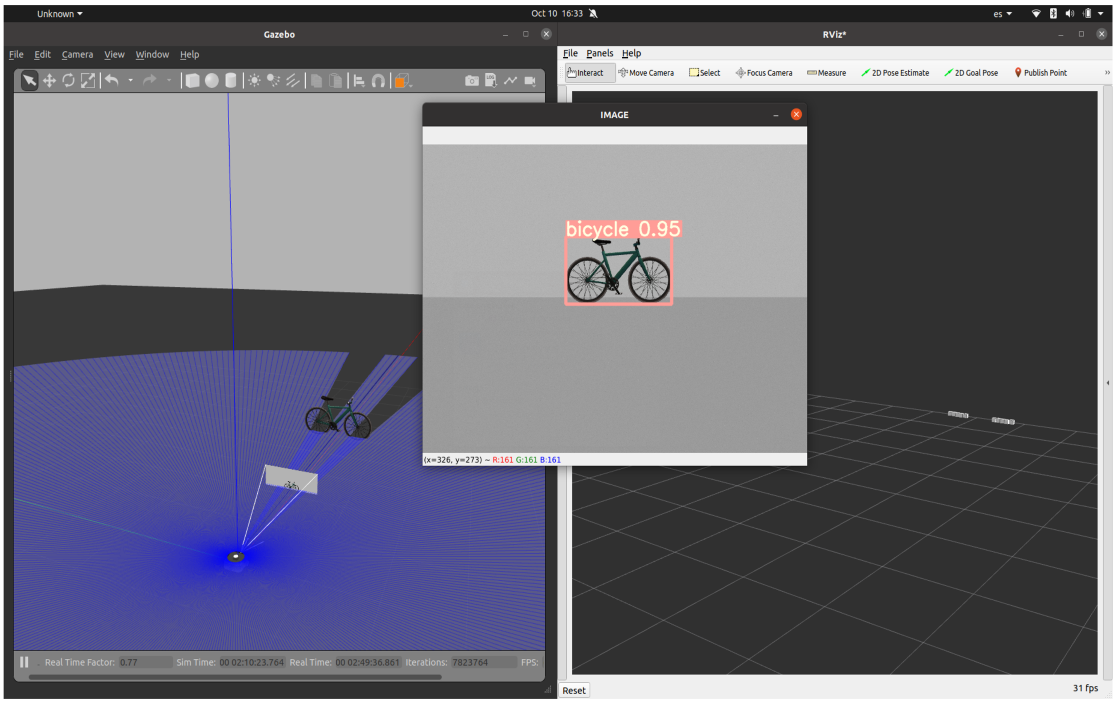


Figure 7.1 : Object (bicycle) tested in RViz.

Test of a fire hydrant in RViz tool. On the left side of Figure 7.2, the view from the Gazebo simulator is displayed and the right screen shows the points detected in RViz.

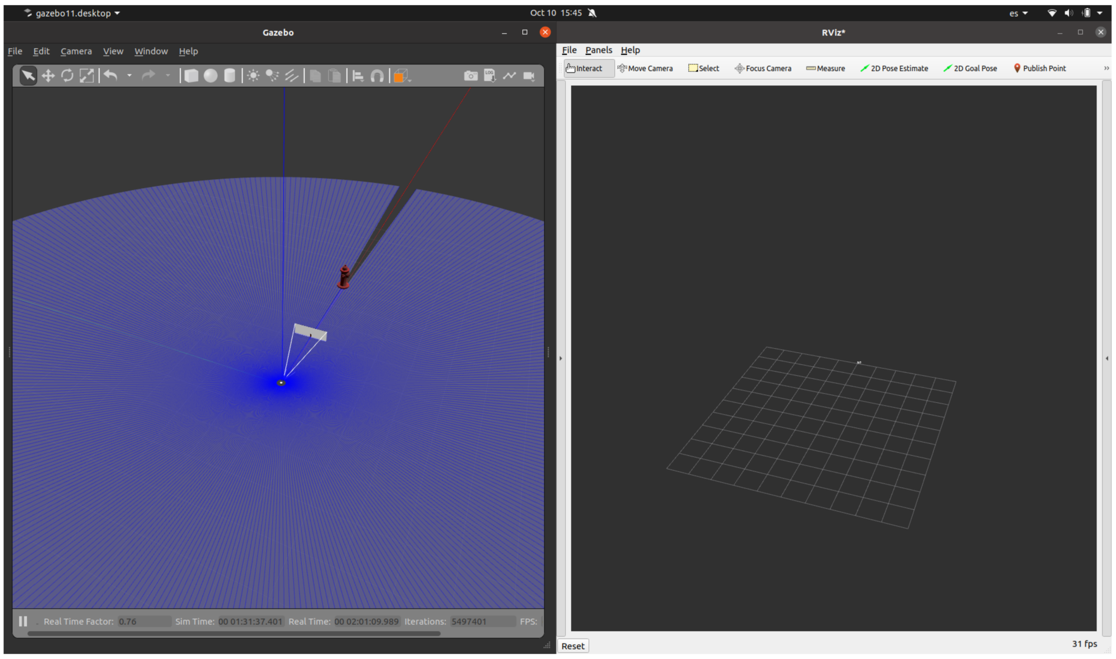

gazebo11.desktop
Oct 10 15:45
Gazebo
File Edit Camera View Window Help
Real Time Factor: 0.76 Sim Time: 00 01:31:37.401 Real Time: 00 02:01:09.989 Iterations: 5497401 FPS:
RViz*
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Goal Pose Publish Point
Reset
31 fps

Figure 7.2: Object (fire hydrant) tested in RViz.

7.2 Appendix B

List of software installed:

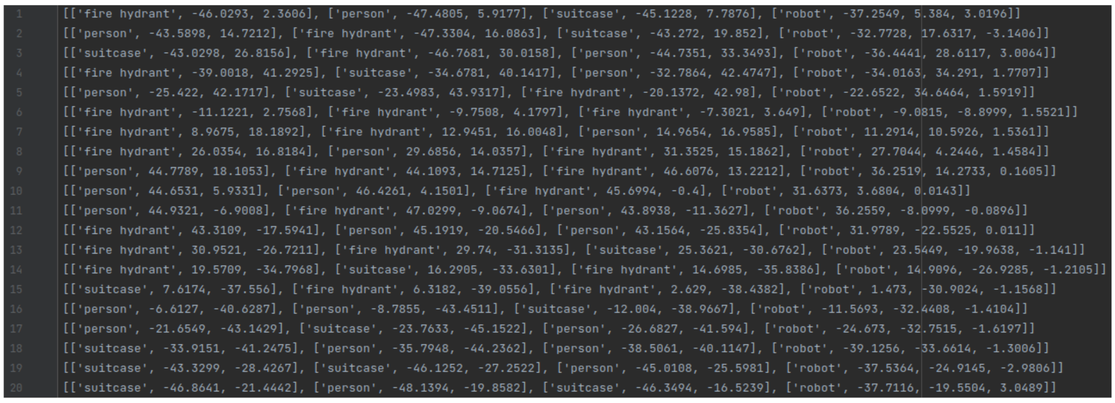
- OS: Ubuntu 20.4
- ROS: ROS2 Foxy
- Conda enviroment: Miniconda3

Name	**Version**	**Build**	**Channel**
cudatoolkit	11.0.3	h88f8997_10	conda-forge
libgcc-ng	11.2.0	h1234567_1	
numpy	1.22.3	py38h99721a1_2	conda-forge
python	3.8.13	h12debd9_0	
pytorch	1.11.0	cpu_py38h39c826d_1	
torch	1.11.0+cu102	pypi_0	pypi
torchvision	0.12.0+cu102	pypi_0	pypi

Table 7.1: List of software installed in Miniconda3

7.3 Appendix C

Landmarks’ Database.

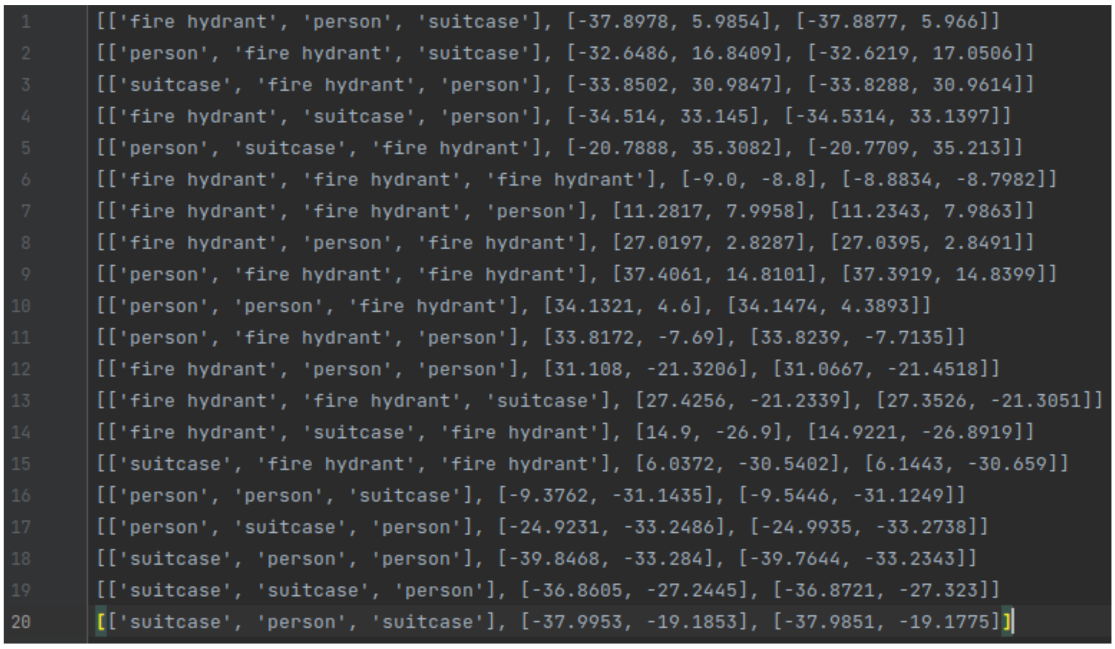


```
[['fire hydrant', -46.0293, 2.3606], ['person', -47.4805, 5.9177], ['suitcase', -45.1228, 7.7876], ['robot', -37.2549, 5.384, 3.0196]]
[['person', -43.5898, 14.7212], ['fire hydrant', -47.3304, 16.0863], ['suitcase', -43.272, 19.852], ['robot', -32.7728, 17.6317, -3.1406]]
[['suitcase', -43.0298, 26.8156], ['fire hydrant', -46.7681, 30.0158], ['person', -44.7351, 33.3493], ['robot', -36.4441, 28.6117, 3.0064]]
[['fire hydrant', -39.0018, 41.2925], ['suitcase', -34.6781, 40.1417], ['person', -32.7864, 42.4747], ['robot', -34.0163, 34.291, 1.7707]]
[['person', -25.422, 42.1717], ['suitcase', -23.4983, 43.9317], ['fire hydrant', -20.1372, 42.98], ['robot', -22.6522, 34.6464, 1.5919]]
[['fire hydrant', -11.1221, 2.7568], ['fire hydrant', -9.7508, 4.1797], ['fire hydrant', -7.3021, 3.649], ['robot', -9.0815, -8.8999, 1.5521]]
[['fire hydrant', 8.9675, 18.1892], ['fire hydrant', 12.9451, 16.0048], ['person', 14.9654, 16.9585], ['robot', 11.2914, 10.5926, 1.5361]]
[['fire hydrant', 26.0354, 16.8184], ['person', 29.6856, 14.0357], ['fire hydrant', 31.3525, 15.1862], ['robot', 27.7044, 4.2446, 1.4584]]
[['person', 44.7789, 18.1053], ['fire hydrant', 44.1093, 14.7125], ['fire hydrant', 46.6076, 13.2212], ['robot', 36.2519, 14.2733, 0.1605]]
[['person', 44.6531, 5.9331], ['person', 46.4261, 4.1501], ['fire hydrant', 45.6994, -0.4], ['robot', 31.6373, 3.6804, 0.0143]]
[['person', 44.9321, -6.9008], ['fire hydrant', 47.0299, -9.0674], ['person', 43.8938, -11.3627], ['robot', 36.2559, -8.0999, -0.0896]]
[['fire hydrant', 43.3109, -17.5941], ['person', 45.1919, -20.5466], ['person', 43.1564, -25.8354], ['robot', 31.9789, -22.5525, 0.011]]
[['fire hydrant', 30.9521, -26.7211], ['fire hydrant', 29.74, -31.3135], ['suitcase', 25.3621, -30.6762], ['robot', 23.5449, -19.9638, -1.141]]
[['fire hydrant', 19.5709, -34.7968], ['suitcase', 16.2905, -33.6301], ['fire hydrant', 14.6985, -35.8386], ['robot', 14.9096, -26.9285, -1.2105]]
[['suitcase', 7.6174, -37.556], ['fire hydrant', 6.3182, -39.0556], ['fire hydrant', 2.629, -38.4382], ['robot', 1.473, -30.9024, -1.1568]]
[['person', -6.6127, -40.6287], ['person', -8.7855, -43.4511], ['suitcase', -12.004, -38.9667], ['robot', -11.5693, -32.4408, -1.4104]]
[['person', -21.6549, -43.1429], ['suitcase', -23.7633, -45.1522], ['person', -26.6827, -41.594], ['robot', -24.673, -32.7515, -1.6197]]
[['suitcase', -33.9151, -41.2475], ['person', -35.7948, -44.2362], ['person', -38.5061, -40.1147], ['robot', -39.1256, -33.6614, -1.3006]]
[['suitcase', -43.3299, -28.4267], ['suitcase', -46.1252, -27.2522], ['person', -45.0108, -25.5981], ['robot', -37.5364, -24.9145, -2.9806]]
[['suitcase', -46.8641, -21.4442], ['person', -48.1394, -19.8582], ['suitcase', -46.3494, -16.5239], ['robot', -37.7116, -19.5504, 3.0489]]
```

Figure 7.3: A text file containing the landmarks' database.

7.4 Appendix D

Robots’ pose computation results.



```
[['fire hydrant', 'person', 'suitcase'], [-37.8978, 5.9854], [-37.8877, 5.966]]
[['person', 'fire hydrant', 'suitcase'], [-32.6486, 16.8409], [-32.6219, 17.0506]]
[['suitcase', 'fire hydrant', 'person'], [-33.8502, 30.9847], [-33.8288, 30.9614]]
[['fire hydrant', 'suitcase', 'person'], [-34.514, 33.145], [-34.5314, 33.1397]]
[['person', 'suitcase', 'fire hydrant'], [-20.7888, 35.3082], [-20.7709, 35.213]]
[['fire hydrant', 'fire hydrant', 'fire hydrant'], [-9.0, -8.8], [-8.8834, -8.7982]]
[['fire hydrant', 'fire hydrant', 'person'], [11.2817, 7.9958], [11.2343, 7.9863]]
[['fire hydrant', 'person', 'fire hydrant'], [27.0197, 2.8287], [27.0395, 2.8491]]
[['person', 'fire hydrant', 'fire hydrant'], [37.4061, 14.8101], [37.3919, 14.8399]]
[['person', 'person', 'fire hydrant'], [34.1321, 4.6], [34.1474, 4.3893]]
[['person', 'fire hydrant', 'person'], [33.8172, -7.69], [33.8239, -7.7135]]
[['fire hydrant', 'person', 'person'], [31.108, -21.3206], [31.0667, -21.4518]]
[['fire hydrant', 'fire hydrant', 'suitcase'], [27.4256, -21.2339], [27.3526, -21.3051]]
[['fire hydrant', 'suitcase', 'fire hydrant'], [14.9, -26.9], [14.9221, -26.8919]]
[['suitcase', 'fire hydrant', 'fire hydrant'], [6.0372, -30.5402], [6.1443, -30.659]]
[['person', 'person', 'suitcase'], [-9.3762, -31.1435], [-9.5446, -31.1249]]
[['person', 'suitcase', 'person'], [-24.9231, -33.2486], [-24.9935, -33.2738]]
[['suitcase', 'person', 'person'], [-39.8468, -33.284], [-39.7644, -33.2343]]
[['suitcase', 'suitcase', 'person'], [-36.8605, -27.2445], [-36.8721, -27.323]]
[['suitcase', 'person', 'suitcase'], [-37.9953, -19.1853], [-37.9851, -19.1775]]
```

Figure 7.4: A text file containing the results of the robot's pose computation results.

References

[1] C. Debeunne and D. Vivet, “A Review of Visual-LiDAR Fusion based Simultaneous Localization and Mapping,” *Sensors*, vol. 20, no. 7, p. 2068, Apr. 2020, doi: https://doi.org/10.3390/s20072068.

[2] I. Loevsky and I. Shimshoni, “Reliable and efficient landmark-based localization for mobile robots,” *Robotics and Autonomous Systems*, vol. 58, no. 5, pp. 520–528, May 2010, doi: https://doi.org/10.1016/j.robot.2010.01.006.

[3] S. Se, D. Lowe, and J. Little, “Mobile Robot Localization and Mapping with Uncertainty using Scale-Invariant Visual Landmarks,” *The International Journal of Robotics Research*, vol. 21, no. 8, pp. 735–758, Aug. 2002, doi: https://doi.org/10.1177/027836402761412467.

[4] W. Cao, L. Zhou, Y. Huang, and A. Knoll, “Autonomous Driving Simulator based on Neurorobotics Platform,” Dec. 2022. [Online]. Available: https://arxiv.org/pdf/2301.00089.pdf

[5] C. -W. Chen, A. -C. Tsai, Y. -H. Zhang and J. -F. Wang, "3D object detection combined with inverse kinematics to achieve robotic arm grasping," 2022 10th International Conference on Orange Technology (ICOT), Shanghai, China, 2022, pp. 1-4, doi: 10.1109/ICOT56925.2022.10008135.

[6] C. M'Sila, R. Ayad and N. Ait-Oufroukh, "Automated Foreign Object Debris Detection System based on UAV," 2022 IEEE International Conference on Networking, Sensing and Control (ICNSC), Shanghai, China, 2022, pp. 1-7, doi: 10.1109/ICNSC55942.2022.10004050.

[7] X. Qu, B. Soheilian, and N. Paparoditis, “Landmark based localization in urban environment,” *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 140, pp. 90–103, Jun. 2018, doi: https://doi.org/10.1016/j.isprsjprs.2017.09.010.

[8] S. Choi, “Triangulation Toolbox: Open-source algorithms and benchmarks for landmark-based localization,” *IEEE Xplore*, May 01, 2014.

[9] H. M. Ebied and M. S. Abdel-Wahab, “Robot Localization Based on Visual Landmarks,” *Proceedings of the Fifth International Conference on Informatics in Control, Automation and Robotics Service*, 2008.

[10] D. C. K. Yuen and B. A. MacDonald, “Vision-based localization algorithm based on landmark matching, triangulation, reconstruction, and comparison,” *IEEE Transactions on Robotics*, vol. 21, no. 2, pp. 217–226, Apr. 2005, doi: https://doi.org/10.1109/tro.2004.835452.

[11] J. S. Esteves, A. Carvalho, and C. Couto, “Generalized geometric triangulation algorithm for mobile robot absolute self-localization,” *IEEE Xplore*, Jun. 01, 2003.

[12] L. Xia *et al.*, “A survey of image semantics-based visual simultaneous localization and mapping: Application-oriented solutions to autonomous navigation of mobile robots,” *International Journal of Advanced Robotic Systems*, vol. 17, no. 3, p. 172988142091918, May 2020, doi: https://doi.org/10.1177/1729881420919185.

[13] D. J. Yeong, G. Velasco-Hernandez, J. Barry, and J. Walsh, “Sensor and Sensor Fusion Technology in Autonomous Vehicles: A Review,” *Sensors*, vol. 21, no. 6, p. 2140, Mar. 2021, doi: https://doi.org/10.3390/s21062140.

[14] H. Zhong, H. Wang, Z. Wu, C. Zhang, Y. Zheng, and T. Tang, “A survey of LiDAR and camera fusion enhancement,” *Procedia Computer Science*, vol. 183, pp. 579–588, 2021, doi: https://doi.org/10.1016/j.procs.2021.02.100.

[15] ——, “Understanding ROS 2 nodes — ROS 2 Documentation: Dashing documentation,”*docs.ros.org*.https://docs.ros.org/en/dashing/Tutorials/Understanding-ROS2-Nodes.html (accessed Feb. 10, 2023).

[16] ——, “Gazebo : Tutorial : Gazebo plugins in ROS,” *classic.gazebosim.org*. https://classic.gazebosim.org/tutorials?tut=ros_gzplugins

[17] ——, “Introduction to URDF — Industrial Training documentation,” *industrial-t raining-master.readthedocs.io*. Available: https://industrial-training- master.readthedocs.io/en/melodic/_source/session3/Intro-to-URDF.html

[18] G. Jocher, “ultralytics/yolov5,” *GitHub*, Aug. 21, 2020. Available: https://github.com/ultralytics/yolov5

[19] R. Xu, H. Lin, K. Lu, L. Cao, and Y. Liu, “A Forest Fire Detection System Based on Ensemble Learning,” *Forests*, vol. 12, no. 2, p. 217, Feb. 2021, doi: https://doi.org/10.3390/f12020217.

[20] T.-Y. Lin *et al.*, “Microsoft COCO: Common Objects in Context,” *arXiv.org*, 2014. https://arxiv.org/abs/1405.0312

[21] ——, “coco128,”*www.kaggle.com*.Available: https://www.kaggle.com/ultralytics/coco128 (accessed Feb. 13, 2023).

[22] ——, “ROBOTIS e-Manual,” *ROBOTIS e-Manual*. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

[23] H. Gavin, “The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems,” *Department of Civil and Environmental Engineering, Duke University,* 2019.

[24] K. Yousif, A. Bab-Hadiashar, and R. Hoseinnezhad, “An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics,” *Intelligent Industrial Systems*, vol. 1, no. 4, pp. 289–311, Nov. 2015, doi: https://doi.org/10.1007/s40903-015-0032-7.

[25] M. Betke and L. Gurvits, “Mobile robot localization using landmarks,” *IEEE Transactions on Robotics and Automation*, vol. 13, no. 2, pp. 251–263, Apr. 1997, doi: https://doi.org/10.1109/70.563647.

[26] G. Grisetti, T. Guadagnino, I. Aloise, M. Colosi, B. Della Corte, and D. Schlegel, “Least Squares Optimization: From Theory to Practice,” *Robotics*, vol. 9, no. 3, p. 51, Sep. 2020, doi: https://doi.org/10.3390/robotics9030051.

[27] ——, “ROBOTIS e-Manual,” *ROBOTIS e-Manual*. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_01/