ROS 2 Robot With SLAM

Technical Report · October 2024

DOI: 10.13140/RG.2.2.15510.56643

2 authors, including:

Joseph Stewart
University of Cape Town
1 PUBLICATION **0** CITATIONS

SEE PROFILE

SLAM with ROS 2 Platform

Presented by:
Joseph Stewart

Prepared for:
Matt Church
Dept. of Electrical and Electronics Engineering
University of Cape Town

Submitted to the Department of Electrical Engineering at the University of Cape Town in partial fulfilment of the academic requirements for a Bachelor of Science degree in Mechatronics

October 22, 2024

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report SLAM with ROS 2 Platform from the work(s) of other people has been attributed and has been cited and referenced. Any section taken from an internet source has been referenced to that source.
3. This report SLAM with ROS 2 Platform is my own work and is in my own words (except where I have attributed it to others).
4. I have not paid a third party to complete my work on my behalf. My use of artificial intelligence software has been limited to minor use cases as seen in the appendix (specify precisely how you used AI to assist with this assignment, and then give examples of the prompts you used in your first appendix).
5. I have not allowed and will not allow anyone to copy my work with the intention of passing it off as his or her own work.
6. I acknowledge that copying someone else's assignment or essay, or part of it, is wrong, and declare that this is my own work.

Word count: 15120

October 22, 2024

Joseph Stewart

Date

Acknowledgments

First and foremost, I extend my deepest appreciation to my supervisor, Matt Church, for his invaluable guidance, expertise, and unwavering support throughout this project. His insights and encouragement have been instrumental in shaping both this work and my growth as a researcher.

My sincere thanks go to the ROS 2 community for their extensive documentation and support forums, which have been crucial in navigating the complexities of the ROS 2 framework.

I would also like to acknowledge my fellow students who have been a source of motivation over the course of this project.

Finally, I am deeply grateful to my family and friends for their unconditional support, understanding, and encouragement throughout my academic journey. Their belief in me has been a constant source of strength and motivation.

This accomplishment would not have been possible without the support of all these individuals. Thank you.

Abstract

> The ability to simultaneously localize a robot and accurately map its surroundings is one of the most fundamental challenges of robotics.
>
> — Sebastian Thrun

This report presents the development and implementation of a Simultaneous Localization and Mapping (SLAM) system using the Robot Operating System 2 (ROS 2) framework. The project aims to convert an existing robotic platform to ROS 2 while implementing SLAM using as few sensors as possible. The algorithms ORB-SLAM3 and LDSO were evaluated on both a Raspberry Pi and a PC platform to determine which platform-algorithm combination would be best suited for real-world applications. ORB-SLAM3 running on a PC achieved superior performance in its accuracy and loop closure capabilities. The system was capable of processing frames at 129.38 FPS which is well suited for real-time systems. Both ORB-SLAM3 and LDSO both showed consistent scale underestimation however ORB-SLAM3 was more stable in its scaling error. The system had a closed-loop motor control that was successfully implemented to aid in precise robot movement. A suite of ROS 2 nodes were used to achieve the requirements for the project which demonstrates ROS 2's capabilities of distributed compute and wireless data transfer. This research contributes to the understanding of implementing practical SLAM solutions in modern robotic systems and provides valuable insights into the performance trade-offs between different SLAM algorithms and hardware configurations.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background to the study

Simultaneous Localization and Mapping (SLAM) is fundamental in robotics and it has evolved significantly since its inception in the 1980s [1]. SLAM allows a robotic platform to map its environment as well as track its position on the map concurrently. This is used in navigation and has applications ranging from self-driving cars to space exploration [2].

The Robot Operating System (ROS) was first released in 2007 and since then it has become the standard for modern robotic development and research [3]. ROS 2 its successor was introduced in 2017 and it addressed many shortcomings of the original framework. Since robotics is a constantly changing landscape technology is always advancing which means that the integration of SLAM algorithms with modern frameworks like ROS 2 is of utmost importance [4].

LiDAR, cameras and inertial measurement units (IMUs) are the most common sensors used for implementing SLAM [5]. Sensor choice has implications on the performance of the SLAM system such as accuracy and cost [6]. Figure 1.1 illustrates the basic concept of SLAM and shows a robot moving through an environment while simultaneously producing a map of the environment it is traversing.

There is a large demand for SLAM in the industrial setting as well as in the hobbyist setting [7]. Having a platform that is capable of SLAM while using the ROS 2 framework is a strong foundation for research and further development [9].

This project aims to take an existing robot platform and integrate a low-cost SLAM solution while also being built around the ROS 2 framework. The project encompasses hardware integration, software development and the implementation of off-the-shelf SLAM solutions. The system needs to be user-friendly and well-documented to lower the barrier to entry and make future development and research on the platform an easy integration [10].

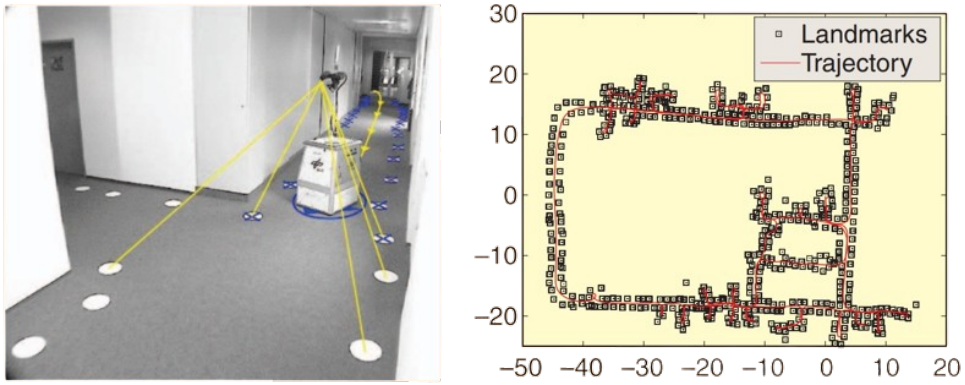


Figure 1.1: Illustration of the fundamental concept of SLAM, showing how a robot moves through an environment while simultaneously estimating its own position and the locations of landmarks. Adapted from [8].

1.2 Objectives of this study

1.2.1 Problems to be investigated

SLAM is a fundamental capability that many modern robotic platforms require for navigation and for interaction with the environment. The development and testing of the associated trade-offs of a ROS 2-based robotic platform capable of performing SLAM is the main objective of this study. The main problems investigated throughout this research project are the challenges of converting an existing robotic platform to ROS 2, including software architecture and driver compatibility; the integration of a minimal sensor suite for SLAM implementation; and the selection and integration of an appropriate off-the-shelf SLAM solution. Further objectives for this study include the analysis of SLAM algorithms and comparing the performance of different solutions. The main performance metrics that will be focused on are the localisation accuracy, map fidelity and the computational efficiency of the systems. Following the selection of the most adequate solution, an extensive user guide will be created for future research and development. This project aims to contribute to the broader understanding of implementing practical SLAM solutions in modern robotic systems using the ROS 2 framework.

1.2.2 Purpose of the study

This project aims to develop a fully functional modern robotic platform capable of performing SLAM running on the ROS 2 operating system. The platform will leverage the capabilities of ROS 2 to take an off-the-shelf SLAM algorithm and create a capable platform.

The purpose of the study is to investigate transitioning current platforms to ROS 2 for SLAM applications. It aims to contribute to existing literature on implementing SLAM for practical use cases. The study intends to provide a comprehensive guide for implementing

SLAM solutions on ROS 2 which can be used with a variety of hardware. It addresses the complexities of sensor integration, software architecture as well as performance optimization. By documenting this process the barrier to entry for developing this platform further in the future is significantly decreased.

1.3 Scope and Limitations

The development of a fully autonomous, multi-purpose robotic platform with advanced SLAM capabilities is outside the scope of this project for many factors such as the complexity of such systems and the limited 13-week timeframe of this project. The implementation of custom SLAM algorithms or the development of novel sensors is beyond the project's scope. Instead the focus is on integrating existing off-the-shelf solutions with ROS 2. The project will be limited to indoor environments and structured settings due to the complexity of outdoor navigation and the need for more sophisticated sensor arrays.

Given the resource limitations and the 13-week time constraint, the scope of the study is outlined as follows:

1. Conversion of an existing robotic platform to ROS 2, focusing on core functionalities
2. Integration of a minimal set of sensors required for basic SLAM operations
3. Implementation of an off-the-shelf SLAM solution compatible with ROS 2
4. Development of necessary ROS 2 nodes and interfaces for sensor data processing and robot control
5. Creation of a basic user interface for robot control and data visualization
6. Performance evaluation of the SLAM implementation in controlled indoor environments
7. Documentation of the development process, including a user guide and source code repository
8. Optimization of the system for real-time performance within the capabilities of the chosen hardware

This scope allows for a comprehensive exploration of ROS 2-based SLAM implementation while remaining achievable within the project's timeframe and resources.

1.4 Plan of development

This report begins with an introduction that provides background information on SLAM and ROS 2. The project objectives and the scope and limitations of the project are all defined. Chapter two presents a comprehensive literature review covering everything relevant to the project. This includes SLAM methodologies, feature detection algorithms, state-of-the-art SLAM implementations and the evolution from ROS to ROS 2. Chapter three details the system design which includes the technical requirements, system architecture, hardware components and software implementation considerations for the system. Chapter four describes the practical implementation of the system. In this chapter, the hardware setup, ROS 2 environment configuration, node implementation and SLAM system integration are discussed in great detail. Chapter five presents and discusses the results from both simulation studies and experimental testing. The results are analyzed and the performance of ORB-SLAM3 and LDSO is evaluated. Finally, Chapter six provides conclusions drawn from the research and offers recommendations for future improvements and development directions.

Chapter 2

Literature Review

2.1 Introduction to Literature Review

This literature review provides an overview of Simultaneous Localization and Mapping (SLAM), a fundamental problem in robotics that forms the backbone for navigation and environment interaction. The review shows SLAM's evolution from the 1980s to its current state, exploring key methodologies and techniques including filter-based and graph-based approaches, with a focus on visual SLAM techniques. State-of-the-art SLAM algorithms such as ORB-SLAM3 and LDSO are examined. The review then explores the integration of SLAM with the Robot Operating System (ROS) as well as ROS 2. This review aims to provide the reader with insights into current technologies and future directions in the field of autonomous robotics.

2.2 Introduction to SLAM

SLAM is a key component in robotics as it enables autonomous navigation and environment interaction. SLAM algorithms allow robots to construct a map of an unknown environment while keeping track of their location within it [1]. This is essential for many applications, including autonomous vehicles, unmanned aerial vehicles (UAVs), service robots, and augmented reality systems [2].

SLAM was first formulated in the 1980s thanks to the work by Smith, Self, and Cheeseman laying the groundwork for probabilistic approaches to the problem [13]. Their work introduced the concept of representing the uncertainty of both the robot's pose and the landmark locations using a joint state vector and covariance matrix. This led to the widespread adoption of this approach in modern SLAM techniques and algorithms.

Since SLAM was formulated it has changed significantly due to advancements in sensor

technology, computational power, and algorithm design [5]. SLAM systems initially were limited to small-scale indoor environments due to the lack of computational resources. As SLAM modernised algorithms can operate in large-scale, dynamic environments, both indoors and outdoors, thanks to improved sensors, more efficient algorithms, and increased computing power. Figure 2.1 shows the advancements in the available SLAM algorithms since its conceptual starting place to the present.

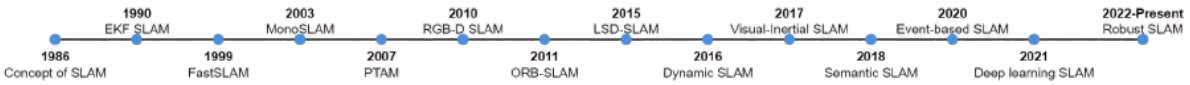


Figure 2.1: Evolution of SLAM (Simultaneous Localization and Mapping) technology from its conceptual introduction in 1986 to present-day robust implementations, highlighting key milestones and algorithmic advancements.

The challenge that SLAM tries to overcome is the interdependence of localization and mapping. To create an accurate map, the robot needs to know its precise location but to determine its location, it needs an accurate map. This problem is further complicated by other factors including sensor noise, environmental dynamics, and computational constraints [14].

2.3 SLAM Methodologies

The type of sensors used in the implementation of SLAM will determine how the SLAM is categorised. This section provides an overview of the main SLAM approaches currently in use.

2.3.1 Filter-based SLAM

Filter-based approaches estimate the robot's pose and map features using filters that have their roots in probabilistic functions. There are two types of filter-based SLAMs, the first is EKF SLAM and the second is particle SLAM.

EKF SLAM, which was introduced by Smith et al. [13], uses a Gaussian distribution to represent both the joint posterior of the robot pose as well as the map features. The state vector includes both the robot's pose and the positions of all the observed landmarks in the environment. The covariance matrix represents the uncertainty in the estimated pose

and observed landmarks with their associated correlations. In the prediction step, the model that represents the robot's motion is used to predict its new pose. In the update step, the sensor measurements are used to correct the predicted pose and update the map. EKF SLAM is very fast and efficient in smaller maps however it suffers from linearization errors [14].

Particle filter SLAM such as FastSLAM [15]instead uses a set of particles to represent the posterior distribution. Within each particle, the robot's pose estimate and a set of landmark estimates are contained. FastSLAM addresses some limitations of EKF SLAM by handling non-linear motion models and non-Gaussian noise. It also offers improved scalability with a computational complexity of O(M log N), where M is the number of particles and N is the number of landmarks [8]. Particle filter SLAM can suffer from particle depletion in large spaces and requires a large number of particles to accurately represent complex posterior distributions.

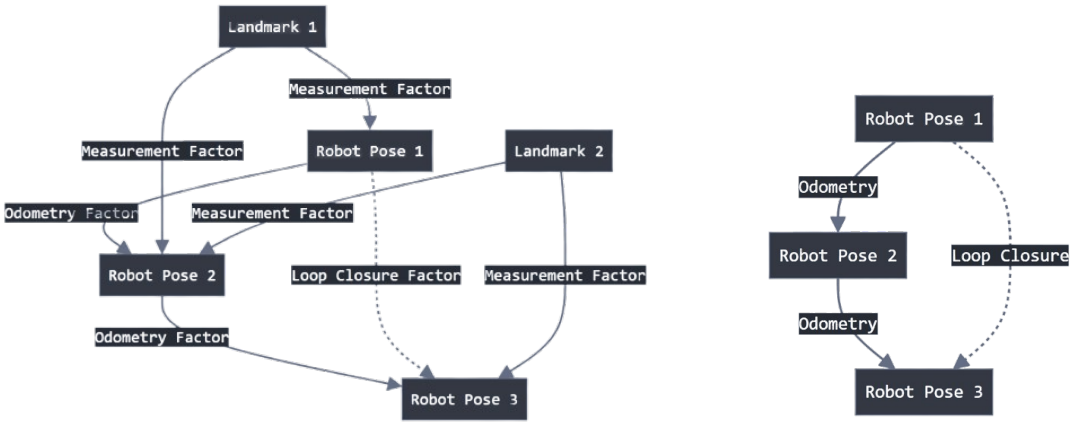
2.3.2 Graph-based SLAM

Graph-based SLAM algorithms have a different way of tackling the issue, they instead turn the problem into a graph and optimise it [16]. Nodes in the graph represent the robot's poses and landmarks whereas the edges represent the constraints [16]. This method can handle large environments in an efficient manner which has led to its mainstream adoption. Within the graph-based approach, there is pose graph SLAM and factor graph SLAM. Figure 2.2 shows how the two graph-based methods differ from each other.

Pose graph SLAM as seen in Figure 2.2(b) tries to optimise the robot's trajectory without relying on the landmarks in the environment. In this type of SLAM, nodes represent the poses and edges represent the pose constraints [17]. Levenberg-Marquardt or the Gauss-Newton method is used for the optimisation of the generated graph. Pose Graph SLAM updates as new measurements arrive, making it suitable for online SLAM [17]. iSAM (incremental Smoothing and Mapping) [50] is an implementation of Pose Graph SLAM as well as Cartographer [51] which is a real-time SLAM system developed by Google.

Factor graph SLAM, which can also be seen in 2.2(a), unlike pose graph SLAM contains measurements and constraints [18]. In a factor graph, variables or nodes represent poses and landmarks, while factors (edges) represent probabilistic constraints on these variables. As shown in Figure 2.2(a) factor graph SLAM contains poses and landmarks as nodes and then coming into the nodes are measurements such as odometry and landmark positions. The advantages that factor graphs have are that they can combine the measurements for multiple dependencies at each pose and its efficient optimisation [18]. GTSAM (Georgia Tech Smoothing and Mapping) [52] contains a toolbox used for optimising factor graphs and SLAM++ [53] uses the topology for efficient optimisation for larger environments.

By comparing Figures 2.2(a) and 2.2(b) is can be observed that factor graph SLAM provides a bigger picture of the environment because it includes landmarks. Pose graph



(a) Factor Graph SLAM (b) Pose Graph SLAM

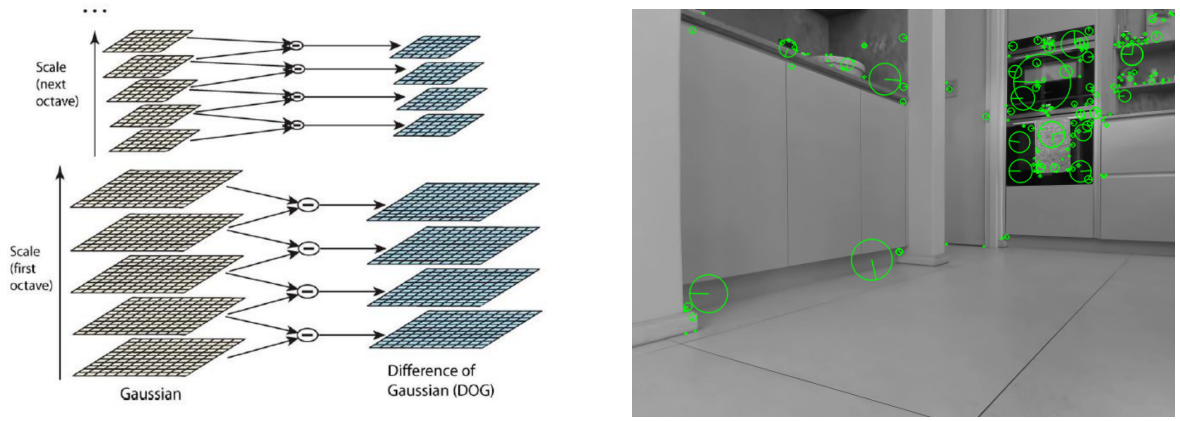
Figure 2.2: Comparison of Graph SLAM methods

SLAM focuses on the robot's trajectory which leads to it being far simpler. The choice between these approaches often depends on the specific requirements of the SLAM system, such as computational constraints.

2.3.3 Visual SLAM

Before discussing visual SLAM, feature detection needs to be introduced as it is fundamental to how many SLAM algorithms work. There are two types of visual SLAM techniques, direct and indirect SLAM. Direct visual SLAM methods operate directly on pixel intensities, minimizing photometric errors between frames to estimate camera motion and scene structure without extracting discrete features. Indirect (or feature-based) visual SLAM methods first extract and match distinct features from images, then use these features to estimate camera poses and build a sparse map of the environment. Three prominent feature detection algorithms used in visual-based SLAMs are reviewed in the following sections followed by a comparison between different types of visual SLAM approaches.

SIFT (Scale-Invariant Feature Transform)



(a) DoG approach to scale-space extrema detection, adapted from [55]

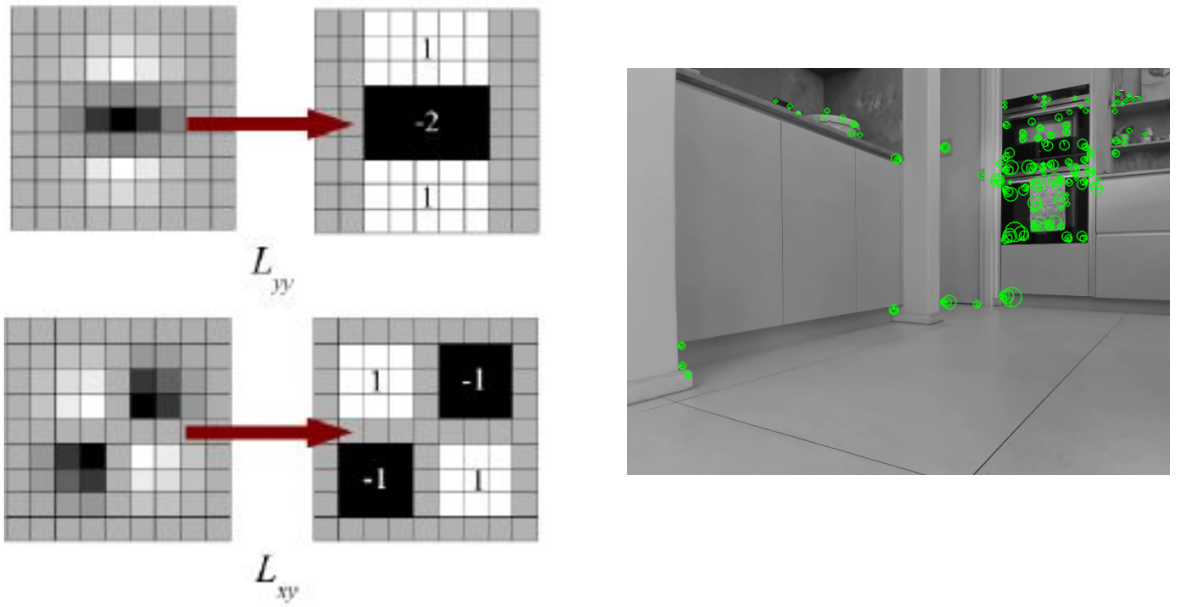
(b) SIFT features detected on a sample image

Figure 2.3: DoG for extrema detection and SIFT features overlay on a still image

SIFT which was initially introduced by Lowe [27] is a robust feature detection algorithm. It is invariant to scale, rotation, and illumination changes which is why it has become a prominent feature detection algorithm. As shown in Figure 2.3(a), SIFT uses a Difference of Gaussians (DoG) approach for scale-space extrema detection. Gaussian blurring is applied at different scales (octaves) and for each scale, a DoG is computed as the difference between two blurred layers. This approach enables SIFT to detect keypoints of various sizes due to it being run at multiple scales. This approach leads to it being accurate but computationally intensive. Figure 2.3(b) shows SIFT features overlayed on a still image. Table 2.1 indicates that SIFT has a moderate number of features (C1 and C2) but at a cost as the matching time is slow (C9). This leads to SIFT being too slow to be run on fast-paced real-time systems.

SURF (Speeded Up Robust Features)

SURF proposed by Bay et al. [29] is designed as a faster alternative to SIFT. Figure 2.4(a) shows how SURF approximates the Laplacian of Gaussian with box filters. This leads to SURF having efficient computation by using these integral images. The top row illustrates the approximation for L_{yy} which is the second-order partial derivative in y. The bottom row shows L_{xy} which is the second-order partial derivative in xy. SURF constructs scale-space representations quickly and computes 64 or 128-dimensional descriptors based on Haar wavelet responses. Figure 2.4(b) shows SURF features overlayed on a still image. SURF's Fast Hessian detector and integral image calculations result in faster feature detection. SURF has faster feature detection times than SIFT as seen in Table 2.1 columns C5 and C6. This makes it more suitable for real-time SLAM applications, especially on platforms with limited computational resources.



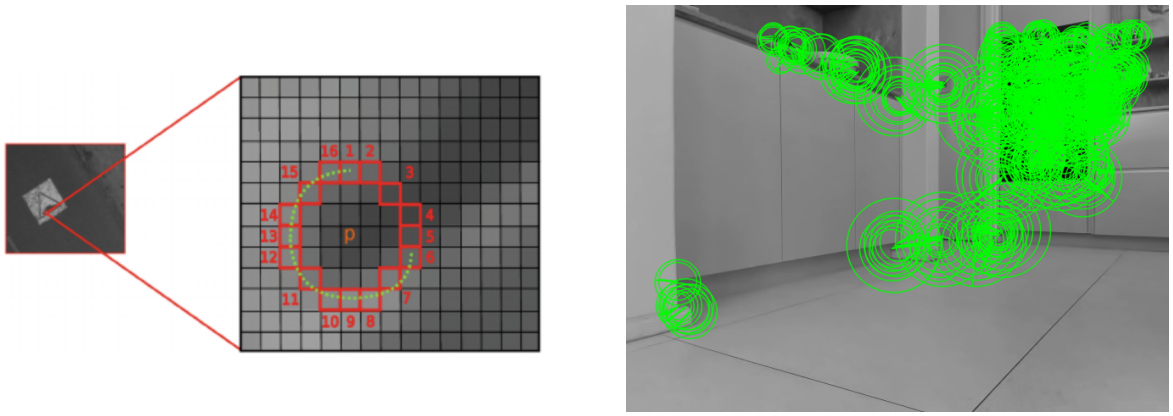
(a) Approximated Laplacian of Gaussian with Box Filter, adapted from [29]

(b) SURF features detected on a sample image

Figure 2.4: LoG approximation with Box Filter and SURF features overlay on a still image

ORB (Oriented FAST and Rotated BRIEF)

ORB which was developed by Rublee et al. [26] combines FAST keypoint detection with a modified BRIEF descriptor. It uses FAST to identify the corner points in an image as illustrated in Figure 2.5(a). A pixel is considered a corner if 12 contiguous pixels in a circular pattern are significantly brighter or darker than the centre. ORB then applies the Harris corner measure and assigns orientation for rotation invariance. For description, the algorithm uses a rotation-aware version of BRIEF which creates binary descriptors through pixel intensity comparisons. Figure 2.5(b) shows ORB features detected on an image. The combination of the FAST corner points and the modified BRIEF descriptor results in a computationally efficient algorithm. This makes it suitable for real-time applications as demonstrated by ORB-SLAM3 [31]. Table 2.1 shows that ORB features outperform SIFT and SURF in terms of detection speed and feature count.



(a) FAST keypoint detector, adapted from [54]

(b) ORB features detected on a sample image

Figure 2.5: FAST keypoint detection and ORB features overlay on a still image

Table 2.1: Quantitative comparison and computational costs of different feature-detector-descriptors. Adapted from [46]

Algorithm	C1	C2	C3	C4	C5	C6	C7	C8	C9
SIFT	3125.7	3662.8	470.8	84.8	0.2827	0.3119	0.5202	0.0064	1.1212
SURF(128D)	4317.8	4744.7	278.3	52.3	0.1847	0.2003	0.7997	0.0087	1.1934
SURF(64D)	4317.8	4744.7	310.2	74.8	0.1806	0.1954	0.4254	0.0078	0.8092
KAZE	1517.5	1660.7	212.8	37.3	0.2902	0.2941	0.1006	0.0055	0.6904
AKAZE	1633.0	1776.0	231.5	33.0	0.0995	0.1013	0.0437	0.0057	0.2502
ORB	9782.7	10828.0	742.2	162.7	0.0385	0.0427	1.0530	0.0068	1.1410
ORB(1000)	955.2	955.7	122.8	21.2	0.0129	0.0133	0.0113	0.0051	0.0426
BRISK	6612.3	7476.8	544.7	113.7	0.1097	0.1253	0.9319	0.0066	1.1735
BRISK(1000)	966.0	958.7	127.5	20.0	0.0200	0.0206	0.0149	0.0054	0.0609

Legend:
C1: Features Detected in the 1st Image
C2: Features Detected in the 2nd Image
C3: Features Matched
C4: Outliers Rejected
C5: Feature Detection & Description Time (s) for 1st Image
C6: Feature Detection & Description Time (s) for 2nd Image
C7: Feature Matching Time (s)
C8: Outlier Rejection & Homography Calculation Time (s)
C9: Total Image Matching Time (s)

Visual SLAM Approaches

Visual SLAM systems use camera images as the input for mapping and localization. These systems have proven their importance due to their lower cost and the ability to

construct 3D maps from the gathered data in the form of point clouds. Visual SLAM contains Monocular SLAM, Stereo SLAM and RGB-D SLAM. Each one has its distinct characteristics and a summary of their differences can be seen in Table 2.2.

Monocular SLAM systems, such as MonoSLAM [19], PTAM [20], and ORB-SLAM [31], use a single camera to estimate the robot's motion and reconstruct the environment. Table 2.2 shows how these systems are more cost-effective and have a simple hardware setup. These systems face challenges such as scale ambiguity in estimation due to the lack of depth data. To resolve this additional information is required as it relies on indirect depth information derived through motion. Monocular SLAM is suitable for both indoor and outdoor environments, particularly those rich in texture.

Stereo SLAM systems utilize two cameras to provide depth information directly. RTAB-Map [21] and S-PTAM [22] are two common implementations of this system. Table 2.2 shows how stereo SLAM offers absolute scale estimation due to the known baseline between cameras. This leads to simplified scale estimation, unlike the problem that plagues monocular systems. Stereo SLAM provides direct depth measurements making it suitable for a variety of indoor and outdoor environments with various textures. However, stereo systems generally have higher computational intensity than monocular systems.

RGB-D SLAM systems use depth sensors along with RGB cameras to create dense 3D maps. KinectFusion [23] and ElasticFusion [45] are state-of-the-art RBG-D SLAM algorithms used in modern applications. Table 2.2 shows how these systems have dense reconstruction and improved performance in texture-poor environments due to the depth sensor. RGB-D SLAM provides direct depth information from the sensor and absolute scale estimation. These systems typically have higher computational complexity and are more suited to indoor environments due to the depth sensors normally having a range of less than 5 meters.

Each of the mentioned visual SLAM approach have their own trade-offs. Factors such as hardware complexity, computational requirements, and environmental suitability vary greatly across the systems. Choosing a method depends on the specific application requirements, with monocular systems offering simplicity and low cost, stereo systems providing a balance of depth accuracy and versatility, and RGB-D systems excelling in dense mapping and operation in texture-poor environments.

Table 2.2: Comparison of Visual SLAM Methods

Factor	**Monocular SLAM**	**Stereo SLAM**	**RGB-D SLAM**
Sensor Type	Single camera	Two cameras	RGB camera + depth sensor
Scale Estimation	Ambiguous, requires additional information	Absolute (known baseline)	Absolute

Continued on next page

Table 2.2 – continued from previous page

Factor	**Monocular SLAM**	**Stereo SLAM**	**RGB-D SLAM**
Environment Suitability	Indoor and outdoor, texture-rich environments	Indoor and outdoor, various textures	Indoor, limited range environments
Depth Information	Indirect, through motion	Direct, from stereo matching	Direct, from depth sensor
Computational Complexity	Lower	Medium	Higher
Key Advantages	Low cost, simple hardware setup	Direct depth measurement, improved robustness, absolute scale	Dense reconstruction, improved performance in texture-poor environments
Example Systems	MonoSLAM [19], PTAM [20], ORB-SLAM [31]	RTAB-Map [21], S-PTAM [22]	KinectFusion [23], ElasticFusion [45]

2.3.4 LiDAR SLAM

LiDAR (Light Detection and Ranging) systems use laser range finders to create accurate 2D/3D point cloud maps. They are particularly useful in outdoor environments and are well-suited for long-range mapping. Cartographer [24] and LOAM [25] are two examples of LiDAR-based SLAMs. LiDAR SLAM offers high accuracy due to precise distance measurements due to the measurements being absolute. This enables these systems to be able to perform large-scale mapping and for them to be robust to lighting conditions. LiDAR SLAM can be computationally expensive especially when using 3D LiDAR sensors and is prone to motion distortion in point clouds. LiDAR also suffers in feature-poor environments as it requires features to get distance measurements and without these measurements, LiDAR SLAM cannot localise itself or build a map. Recent advancements in LiDAR SLAM focus on real-time performance, multi-sensor fusion, and semantic mapping.

2.4 SLAM Algorithms

In this section two prominent SLAM algorithms are explored.The SLAM systems are ORB-SLAM3 and LDSO. The monocular versions of these two algorithms will be the primary focus as it is the type of SLAM that will be implemented due to its advantages listed in Table 2.2. These algorithms have different approaches to dealing with visual

SLAM and by examining these algorithms insight into the current state-of-the-art in visual SLAM technology can be gained.

2.4.1 ORB-SLAM3

ORB-SLAM3 which was developed by Campos et al. [31] is a complicated visual SLAM algorithm that tracks the camera's position. It simultaneously processes camera input to create and maintain a map of the environment. Figure 2.6 shows how ORB-SLAM3 works. With each new input frame, ORB features are first extracted. These distinctive visual elements are then used in the tracking component which forms the backbone for the algorithm. Tracking matches the extracted features with the existing map to determine the camera position and orientation. With each new frame, the system decides if the frame should be designated as a keyframe. Frames that contain significant new information about the environment are designated as a keyframe.

Each time a new keyframe is created it undergoes local mapping. This involves creating new map points as well as refining existing ones. Following that it performs local bundle adjustment to optimize the recent portion of the map. The system performs loop detection following that which involves continuously checking if the current view matches any previously mapped areas. If a loop is detected the loop closing component corrects accumulated drift. If a loop is found global map optimization refines the entire map to ensure overall consistency and alignment. The map database stores all map points and keyframes, interacting with tracking, local mapping, and loop closing components to provide and update map information.

The flowchart in Figure 2.6 illustrates several feedback loops in the system: tracking continuously uses and updates the map database, and after local mapping or loop closing, the system returns to tracking with an improved map. This cyclical process allows ORB-SLAM3 to build and maintain an accurate 3D map of the environment while robustly tracking the camera's position, even in challenging scenarios like revisiting previously mapped areas or operating in large-scale environments. The system's design enables real-time operation, with multiple components running in parallel, making it efficient and effective for a wide range of applications. As demonstrated by Campos et al. [31], ORB-SLAM3 achieves high accuracy in both trajectory estimation and mapping while maintaining real-time performance on standard CPUs. The algorithm has demonstrated superior performance on various datasets, including EuRoC, TUM-VI, and KITTI. It achieves high accuracy in both trajectory estimation and mapping while maintaining real-time performance (Table II in [31]).

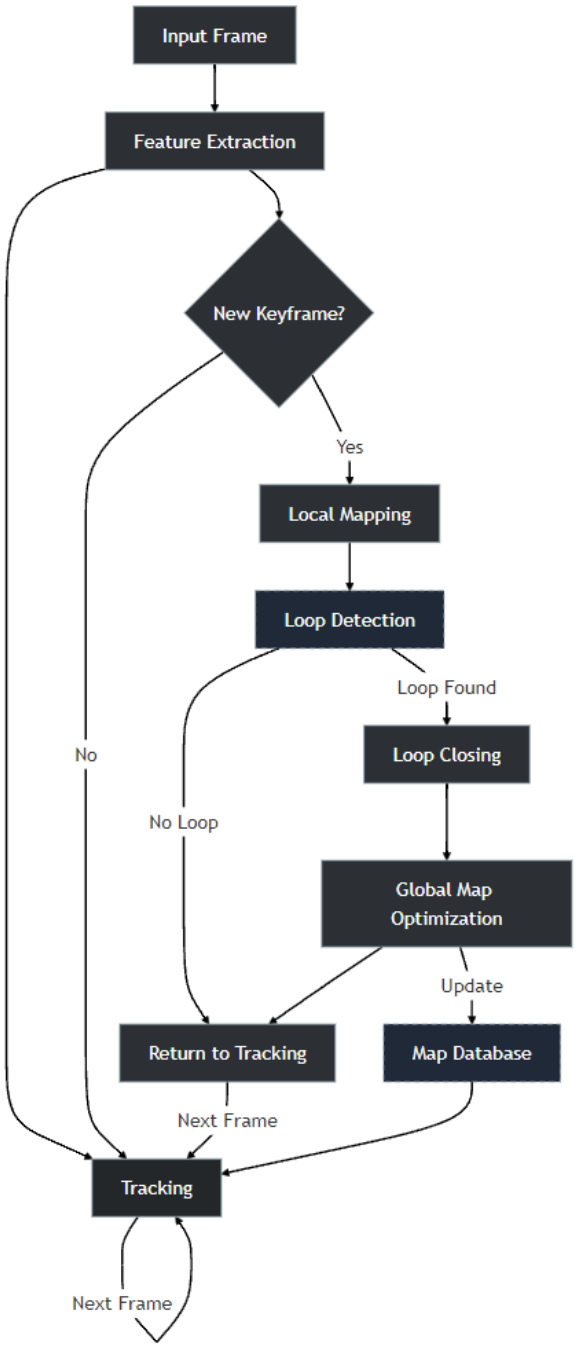


Figure 2.6: Flowchart of ORB-SLAM3 illustrating the main components and data flow in the monocular configuration.

2.4.2 LDSO: Loop Closure Direct Sparse Odometry

LDSO (Loop closure Direct Sparse Odometry) was developed by Gao et al. [47]. It is an extension of the Direct Sparse Odometry (DSO) framework. DSO which was introduced by Engel et al. [40] is a visual odometry method. DSO operates directly on image intensities rather than extracting and matching features. It minimizes a photometric error over a sliding window of recent frames which allows it to estimate camera motion and scene

structure. DSO's direct approach allows it to work well in environments with poor textures which is where feature-based methods might struggle. DSO lacks loop closure capabilities and this can lead to drift over long trajectories.

LDSO addresses this limitation of DSO by implementing loop closure detection and pose graph optimization. DSO focuses solely on local accuracy through its sliding window optimization. LDSO on the other hand introduces a global pose graph. This addition aids in maintaining consistency over the entire trajectory. The key innovation of LDSO lies in its hybrid approach that utilises both direct and indirect techniques. LDSO retains the benefits of direct methods from DSO and builds on them by using a point selection strategy that favours corner features that are repeated. This approach enables the system to perform loop closure detection using traditional feature-based techniques and maintains robustness in feature-poor environments [47].

To understand how LDSO works Figure 2.7 explains the steps LDSO employs in its algorithm. With each new image frame, LDSO performs hybrid point selection. It chooses points that are both good for direct alignment and feature matching. These points are then used for direct image alignment which allows the algorithm to estimate the camera's movement between frames. Local optimization is then performed within a sliding window of recent frames which refines the camera trajectory. The system then updates its pose graph (overall camera path). LDSO then attempts to detect loop closures using a Bag of Words (BoW) approach, checking if the camera has returned to a previously visited location. If a loop closure is detected LDSO performs global pose graph optimization to correct accumulated drift. Then the global map is updated and the process repeats for the next frame. This pipeline allows LDSO to combine the advantages of direct and indirect methods which results in a more consistent and accurate reconstruction.

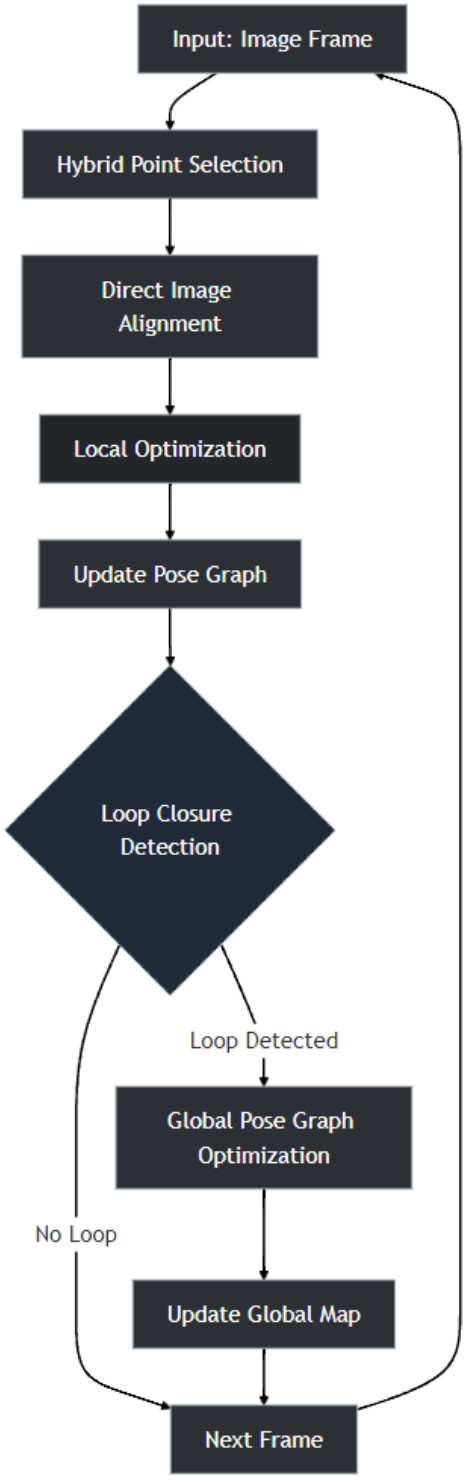


Figure 2.7: Flowchart of LDSO illustrating the main components and data flow in the monocular configuration.

2.4.3 Visual-Based SLAM Comparison

In a comprehensive evaluation by Safarova et al. [57] ORB-SLAM3 consistently outperformed LDSO. In the EuRoC MH01 (mono) dataset, ORB-SLAM3 achieved an average RMS ATE of 0.016m which is significantly lower than LDSO's 0.054m. Across all the EuRoC datasets ORB-SLAM3 had an average RMS ATE 6.83 times smaller than that of DSO's. ORB-SLAM3's multi-map capabilities, visual-inertial odometry, and global bundle adjustment contribute to its higher accuracy [31]. LDSO is a significant step forward in direct SLAM methods and bridges the gap between the robustness of direct methods and the loop closure capabilities of feature-based approaches. ORB-SLAM3 remains the superior choice for state-of-the-art monocular SLAM performance. Its ability to maintain long-term consistency, handle multi-session operations, and provide superior accuracy makes it a more suitable option for a wide range of applications.

2.5 Robot Operating System (ROS) and ROS 2

2.5.1 Introduction to ROS

In 2007 ROS which is an open-source middleware framework for robotics applications was released to the public. It contains tools, libraries, and conventions that simplify the task of creating complex robotic platforms [3]. ROS is built on a distributed computing model which is centred around nodes, topics, and messages. Nodes are individual processes that perform computations that can communicate with each other over topics. Topics are named buses over which nodes exchange messages which are typed data structures. This type of architecture has a high degree of modularity and allows complex systems to be built from reusable components.

ROS has comprehensive package management for organizing and sharing code. ROS packages are fundamental to software organization. Packages contain the nodes, libraries, datasets and configuration files to perform a specific set of actions. These packages can be easily shared and reused across different robotic systems. This leads to code modularity and reduces development time. The introduction of standardized libraries, protocols, and the package system in ROS was a first for robotics research. ROS provides a common platform for researchers and developers to share and build upon current work. This allows rapid prototyping and code reuse to occur which has resulted in ROS becoming a standard in robotics research as well as development. ROS has a large active community that continually contributes to its development and expansion [3].

2.5.2 ROS 2: Next Generation ROS

ROS 2 was introduced in 2017 and with it came a significant evolution from its predecessor. Figure 2.8 illustrates the differences in how ROS 2 is different from ROS and highlights the key improvements in the newer version.

Figure 2.8 shows that ROS 2 eliminates the centralized Master node present in ROS 1 in the application layer. This caters for a more distributed and robust system architecture. The middleware layer also has some overhauls in the newer version as the Data Distribution Service (DDS) was introduced. The DDS standardised the communication protocol which enhances real-time performance. At the OS layer ROS 2 has expanded compatibility to include Windows, macOS and real-time operating systems [4].

Figure 2.8 also shows additional features in ROS 2. Enhanced security through the DDS's built-in security and improved Quality of Service are two prominent additions. Support for small embedded systems and multi-robot coordination are two other improvements. These improvements address the evolving needs of the robotics community. The adoption of the DDS has visualized the middleware layer. ROS 2 has successfully addressed many limitations of ROS particularly in terms of real-time performance and scalability.

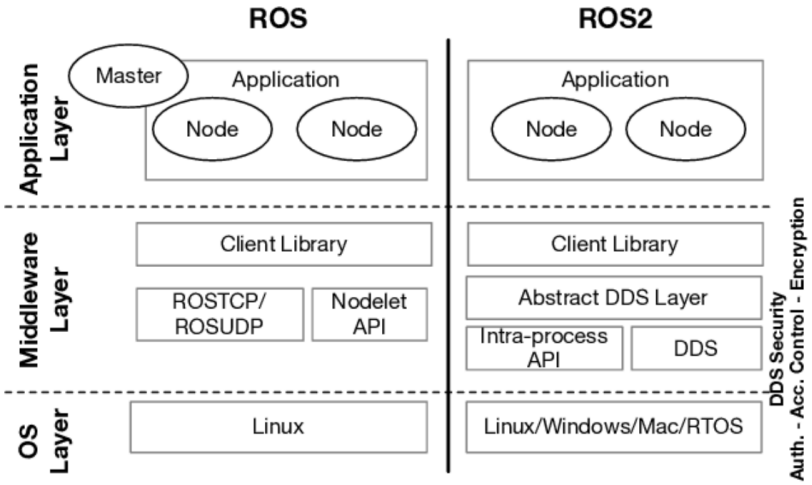


Figure 2.8: ROS/ROS2 architecture overview, adapted from [58]

2.5.3 SLAM in ROS and ROS 2

ROS and ROS 2 both provide packages and tools for implementing SLAM algorithms. ROS 2 has many new possibilities for SLAM implementations due to its improvements. Some popular SLAM packages in the ROS ecosystem include gmapping (ROS 1) [32], cartographer (ROS 1 and ROS 2) [24], rtabmap (ROS 1 and ROS 2) [21], and slam_toolbox (ROS 2) [33].

The integration of SLAM algorithms with ROS 2 is ongoing and is a current task. Currently

existing SLAM packages are being ported to ROS 2 and new ones are being developed that take advantage of ROS 2's improved features [34].

2.6 Conclusion

This literature review has provided an extensive overview of SLAM methodologies and feature detection algorithms. It has also introduced two state-of-the-art SLAM algorithms, that being ORB-SLAM3 and LDSO. It then explored ROS and ROS2 and how it has been used to simplify the devlopment stage as well as change the way researches create robotic platforms. The SLAM field continues to evolve rapidly and it is being driven by advancements in sensor technology and algorithm design. The transition from ROS to ROS 2 offers new possibilities for real-time and scalable SLAM solutions to be implemented. As research in this field progresses more sophisticated SLAM solutions are expected to be developed.

Chapter 3

System Design

This section outlines the design of the SLAM system implemented on a ROS 2 platform. The design builds upon the theoretical foundations discussed in Chapter 2 and addresses the project objectives outlined in Section 1.2.

3.1 Technical Requirements

Before delving into the specifics of the system architecture and the system design, it is crucial to establish the technical requirements that guided the development process. Table 3.1 outlines these requirements, which serve as the foundation for the design decisions detailed in this chapter.

Table 3.1: Technical Requirements

Technical Requirements	**Explanation**
TR1	Development of a ROS 2-based robotic platform capable of SLAM.
TR2	Integration of minimum required sensors (e.g., LiDAR, camera) for SLAM implementation.
TR3	Implementation of an off-the-shelf SLAM solution compatible with ROS 2.
TR4	Creation of a user-friendly interface for robot control and data visualization.
TR5	Development of necessary ROS 2 nodes for sensor data processing and robot control.
TR6	Performance evaluation of the SLAM system in various indoor environments.
TR7	Creation of comprehensive documentation and user guide for the platform.

Continued on next page

Table 3.1 – *Continued from previous page*

Technical Requirements	**Explanation**
TR8	Optimization of the system for real-time performance within hardware capabilities.

These technical requirements informed the design decisions made throughout the development process, ensuring that the final system meets the project's objectives while adhering to the use of ROS 2 and the selected hardware platform. The following sections detail how each of these requirements was addressed in the system architecture and component designs.

3.2 System Architecture

The proposed system architecture seen in Figure 3.1 integrates hardware components, the ROS 2 framework and SLAM algorithms to create a robust and modular robotic platform capable of autonomous navigation and mapping.

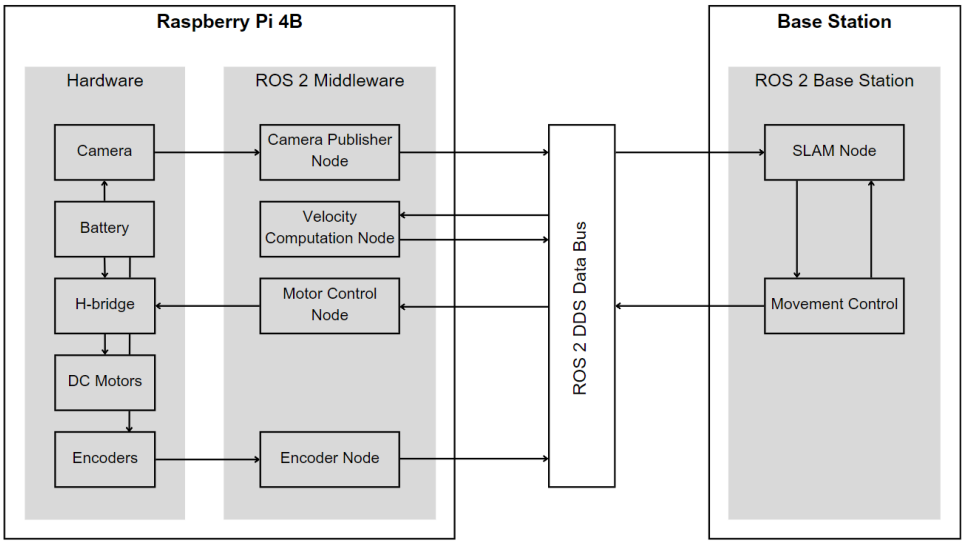


Figure 3.1: High-level system architecture of the ROS 2 Platform with SLAM.

The system architecture consists of three main layers. The Hardware Layer which includes all the physical components such as the camera, encoders, motors and an h-bridge. The ROS 2 Middleware layer provides communication infrastructure and node management between the Raspberry Pi and the base station. Lastly the Base Station which runs SLAM on the data from the robot and is also used to control it.

3.3 Hardware Design

The hardware design focuses on selecting components that meet the project requirements cost-effectiveness and compatibility with ROS 2.

3.3.1 Robotic Platform

For this project, a differential drive robot platform was used. The inherent simplicity and maneuverability of these platforms are well-suited for the SLAM use case. The platform consists of two independently driven wheels and a steel ball castor wheel for stability. This approach ensures the wheeled robot has a simple kinematic model and can be controlled easily. The inverse kinematics of the robot are crucial for converting desired linear and angular velocities into individual wheel speeds.

For a differential drive robot, the kinematic model is described by the following equations:

$$v = \frac{v_r + v_l}{2} \tag{3.1}$$

$$\omega = \frac{v_r - v_l}{L} \tag{3.2}$$

Where v is the linear velocity of the robot, ω is the angular velocity of the robot, v_r is the velocity of the right wheel, v_l is the velocity of the left wheel, and L is the distance between the wheels (wheel base).

The inverse kinematics, which convert desired robot velocities to wheel velocities, are given by:

$$v_l = v - \frac{\omega L}{2} \tag{3.3}$$

$$v_r = v + \frac{\omega L}{2} \tag{3.4}$$

3.3.2 Sensor Selection

A monocular SLAM solution was chosen based on the trade-offs seen in Table 2.2. Monocular SLAM is a cost-effective solution and has great performance hence why it was

chosen. Any camera will work and many camera sensors will be used in this project to compare performance. The platform will be capable of running SLAM on the robot as well as offloading the SLAM to a base station if the robot's computing is not sufficient or the performance is degraded. In the case that the images need to be offloaded to the base station for SLAM to be run externally from the robot the captured images are sent to the base station over the ROS 2 DDS data bus in the camera node as seen in Figure 3.1. Wheel encoders were integrated to provide odometry information. The resolution of the encoders (R) is crucial for accurate velocity estimation and is given by:

$$R = \frac{2\pi}{N} \tag{3.5}$$

where N is the number of pulses per revolution of the encoder.

3.3.3 Computational Hardware

A Raspberry Pi 4 Model B was chosen as the main computational unit due to its processing power and compatibility with ROS 2. The system is capable of handling core ROS 2 functionalities and basic SLAM operations. In the case that the system does not have the resources to do the SLAM computations, the system design incorporates a contingency plan. This plan will involve offloading computationally intensive SLAM tasks to a more powerful base station. This is possible due to ROS 2's distributed architecture for seamless communication between the robot and a remote computer over a common wireless network.

3.4 Software Design

The software architecture is built around the ROS 2 framework, leveraging its improved features over ROS 1 as discussed in Section 2.5.2 of the literature review.

3.4.1 ROS 2 Node Architecture

The system is designed with modularity in mind and it separates the functionalities into distinct ROS 2 nodes. These include the camera node, encoder node, motor control node, SLAM node, and velocity computation node. ROS 2's publish-subscribe model allows for easy communication between these nodes. The key topics include/camera/image_raw/compressed for compressed image data, /wheel_velocities for desired wheel velocities and /encoder_ticks for odometry information.

3.4.2 SLAM System

The goal of this project is to implement an effective SLAM and to do that multiple SLAM systems were implemented and evaluated. This allows the algorithms to be quantitatively compared and the claims made in the literature regarding the performance of different SLAM algorithms to be validated. By implementing and comparing multiple algorithms the aim is to identify the most suitable SLAM solution for this project's specific requirements.

In the following subsections, the two SLAM systems that are used will be discussed in detail and the designs for both SLAM algorithms will be explained in further detail. These subsections are designed to dive deeper into the introductions to the algorithms in the literature review.

ORB-SLAM3

ORB-SLAM3 is a state-of-the-art visual SLAM system that builds upon its predecessors ORB-SLAM and ORB-SLAM2. It is an open-source SLAM algorithm that supports monocular, stereo and RGB-D methods. The system is accurate and robust which makes it a good fit for this project's requirements.

Figure 3.2 provides an overview of the ORB-SLAM3 system architecture. The fundamentals of how ORB-SLAM3 works were highlighted in the literature review however to keep the literature review more of an overview the low-level design of the algorithm was not included.

The Atlas is the multi-map representation of the environment that includes both active and non-active maps. Each map contains map points, keyframes, a covisibility graph and a spanning tree. The Bag of Words (BoW) library DBoW2 is used for the keyframe database and it stores the visual vocabulary and recognition database used for place recognition and loop closure detection. The local mapping thread is responsible for keyframe insertion, map point culling and creation, local bundle adjustment (BA), IMU initialization and scale refinement. The loop & map merging module handles loop closure detection, map merging and essential graph optimization. The module is also responsible for place recognition and loop fusion. A separate thread performs full BA to refine the entire map after significant changes have occurred. The system has a modular design and this allows for efficient parallel processing.

ORB-SLAM3 utilizes ORB features for visual tracking and mapping and this forms the foundation of the methods used within the algorithm. These features are extracted and matched across frames which enables real-time performance. The system uses a keyframe-based approach such that only selected frames are used for mapping and optimization which reduces the computational complexity but importantly maintains accuracy.

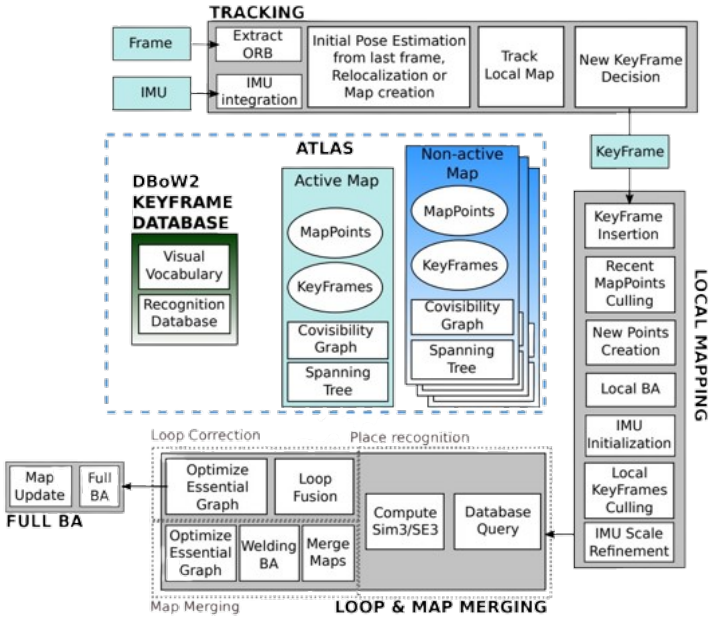


Figure 3.2: High-level system architecture of ORB-SLAM3, adapted from [31].

The monocular SLAM functionality operates by initializing a map from the first few frames and then proceeds to simultaneously track the camera motion and expand the map. The system estimates the camera pose $T_i \in SE(3)$ for each frame i, along with a set of 3D map points $X = \{x_0, ..., x_{l-1}\}$. Here, $SE(3)$ represents the Special Euclidean group in 3D, which describes both rotation and translation in 3D space. Pose estimation is performed by minimizing the reprojection error given in Equation 3.6 which was adapted from [31]. Reprojection error measures the difference between where a 3D point should appear in the image based on its estimated position, and where it appears in the image. The pose estimation is performed by minimizing the reprojection error:

$$\min_{T_i, X} \sum_{j=0}^{l-1} \sum_{i \in K_j} \rho_{Hub}(\|r_{ij}\|_{\Sigma_{ij}^{-1}}) \tag{3.6}$$

where r_{ij} is the reprojection error of point j in frame i, K_j is the set of keyframes observing point j, and ρ_{Hub} is the robust Huber cost function.

ORB-SLAM3 implements three levels of data association to enhance accuracy and robustness:

1. Short-term data association: Tracking features between consecutive frames.
2. Mid-term data association: Matching features with recently mapped points.
3. Long-term data association: Detecting loop closures and relocalization.

The system's map is represented as a covisibility graph, where keyframes are nodes connected by edges indicating shared map points. This structure allows for efficient local

mapping and loop closing operations. The Atlas framework in ORB-SLAM3 enables multi-map and multi-session capabilities, allowing the system to handle tracking losses and merge multiple mapping sessions.

For loop closure detection, ORB-SLAM3 employs an improved place recognition algorithm based on bag-of-words. When a loop is detected, the system performs pose-graph optimization to correct accumulated drift:

$$\min_{\xi} \sum_{(i,j)\in\mathcal{C}} \|log(T_{ij}^{-1} \exp(\xi_i^{-1}\xi_j))\|_{\Sigma_{ij}}^2 \tag{3.7}$$

This equation represents the pose graph optimization in SLAM. Here, ξ_i is the corrected pose for keyframe i, T_{ij} is the relative pose constraint between keyframes i and j, and $\mathcal{C}$ is the set of loop closure constraints. The equation aims to minimize the difference between observed relative poses (T_{ij}) and those computed from corrected keyframe poses ($\exp(\xi_i^{-1}\xi_j)$). This optimization adjusts camera poses to be consistent with loop closures, reducing trajectory drift and improving map accuracy. The result is a set of corrected poses that maintain global consistency in the SLAM system, particularly when revisiting mapped areas or closing large loops.

ORB-SLAM3's monocular implementation has demonstrated excellent performance in various datasets, achieving high accuracy and robustness. The system's ability to reuse maps and perform multi-session SLAM makes it particularly suitable for long-term operation in dynamic environments.

LDSO

This project also implements LDSO [47]. LDSO extends the DSO framework to include loop closure detection and pose graph optimization which enables it to have long-term consistent mapping and localization.

Figure 3.3 illustrates the architecture of the LDSO system. The system combines DSO's windowed optimization approach (shown in blue) with a global pose graph optimization (shown in red). The sliding window maintains 5-7 active most recent keyframes while the marginalized keyframes are added to the pose graph. 3D points in the environment are tracked within the sliding window. When a loop closure candidate is detected within the current frame from the keyframe database a Sim(3) constraint is computed between the current frame and the candidate. The Sim(3) constraint is a similarity transformation that captures both the rotation and translation between frames as well as a scale factor which allows for more accurate loop closure. This constraint is then incorporated into the global pose graph and this allows for correction of accumulated drift. The integration of local optimization with global pose graph refinement is what leads to LDSO maintaining the efficiency of DSO while achieving long-term consistency in mapping and localization.

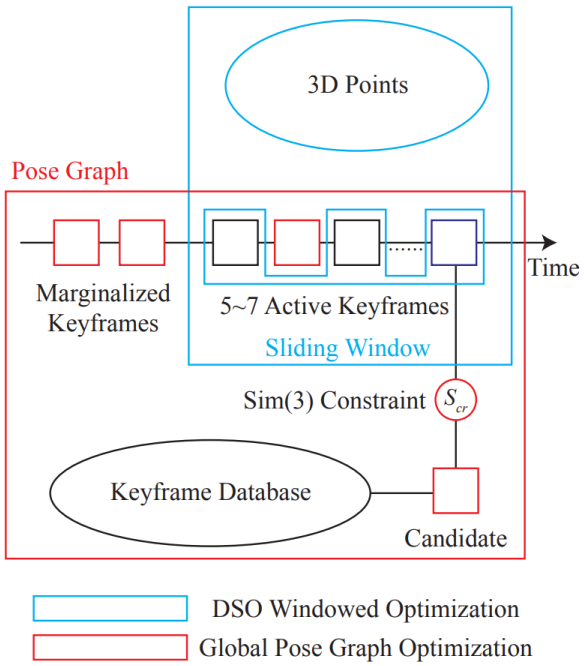


Figure 3.3: LDSO framework, adapted from [47].

LDSO adapts DSO's point selection strategy to favor repeatable corner features but it retains the robustness in feature-poor environments. This is achieved by selecting a portion of pixels as corners using the Shi-Tomasi score [48]. Following that the ORB descriptors [26] for the selected corners are computed while retaining DSO's original selection method for non-corner pixels. This hybrid approach allows the system to maintain DSO's robustness while enabling feature matching for loop closure detection by implementing this parallel indirect visual SLAM method. Figure 3.4 illustrates the difference in point selection between DSO and LDSO.

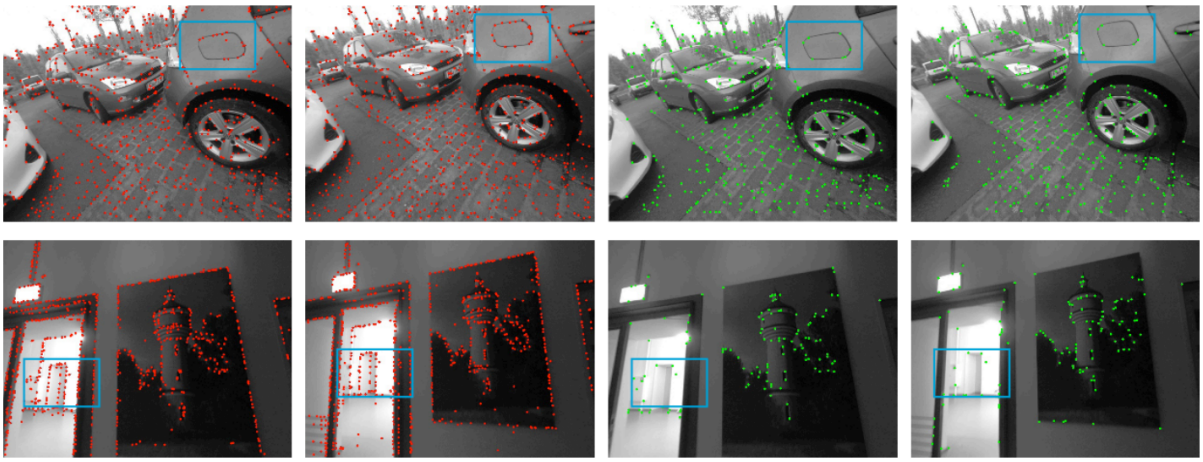


Figure 3.4: Comparison between DSO and LDSO feature selection, adapted from [47].

LDSO similarly to ORB-SLAM3 uses a BoW database (DBoW3) [49] with ORB descriptors from keyframes to implement the loop-closing technique. Loop candidates are found by continuously querying the BoW database. LDSO then matches ORB features between the current keyframe and loop candidates. Following this the system performs RANSAC PnP

to compute an initial SE(3) transformation estimate and optimizes a Sim(3) transformation using both 3D and 2D geometric constraints.

RANSAC PnP (Random Sample Consensus Perspective-n-Point) is a robust method for estimating the camera pose from 2D-3D point correspondences, which efficiently handles outliers in the matching process. SE(3) (Special Euclidean group in 3D), which was also used in ORB-SLAM3 represents rigid body transformations, including both rotation and translation, in 3D space.

The Sim(3) transformation is estimated by minimizing the following cost function which is adapted from [47]:

$$\begin{aligned} E_{loop} = & \sum_{\mathbf{q}i \in Q_1} w_1 \left| \mathbf{S}_{cr} \Pi^{-1}(\mathbf{p}_i, d_{\mathbf{p}_i}) - \Pi^{-1}(\mathbf{q}_i, d_{\mathbf{q}_i}) \right|_2 + \\ & \sum_{\mathbf{q}j \in Q_2} w_2 \left| \Pi(\mathbf{S}_{cr} \Pi^{-1}(\mathbf{p}_j, d_{\mathbf{p}_j})) - \mathbf{q}_j \right|_2 \end{aligned} \tag{3.8}$$

Where S_{cr} is the Sim(3) transformation from the loop candidate (reference) to the current keyframe. This equation combines two error terms to optimize the transformation. The first term minimizes the 3D geometric error for features without depth information in the current frame, while the second term minimizes the 2D reprojection error for features with known depth in the current frame. Q_1 and Q_2 represent sets of matched features without and with depth information respectively. The weights w_1 and w_2 balance the contribution of each term. Π and Π^{-1} denote projection and back-projection functions, while $\mathbf{p}$ and $\mathbf{q}$ represent features in the reference and current frames. By minimizing this cost function, LDSO achieves accurate loop closure by considering both 3D and 2D geometric constraints.

LDSO maintains a global pose graph alongside DSO's sliding window optimization. The pose graph is constructed by extracting relative pose transformations between keyframes from DSO's sliding window, approximating constraints within the marginalization window as pairwise relative pose observations, and adding Sim(3) constraints between loop closure candidates and the current keyframe.

The pose graph is optimized using g2o [17], a graph optimization library. To prevent disturbance to the local windowed optimization, LDSO fixes the current frame's pose estimation during pose graph optimization and does not update the global poses of keyframes in the current window after optimization.

LDSO carefully integrates the loop closure and pose graph optimization with DSO's frontend to maintain the system's overall performance. The VO frontend uses both corner and non-corner points for camera tracking. Loop closure detection and pose graph optimization run in separate threads to minimize the impact on the main tracking thread. The system retains DSO's photometric error minimization for local windowed optimization.

This design allows LDSO to significantly reduce accumulated drift in rotation, translation, and scale while maintaining the robustness and accuracy of DSO's odometry frontend.

3.4.3 Closed-Loop Motor Control

The closed-loop controller for the differential drive robot was designed to provide precise control over the robot's movement, particularly when moving in a straight line. The controller utilizes encoder feedback to continuously adjust the motor speeds which will ensure the robot has accurate and consistent motion.

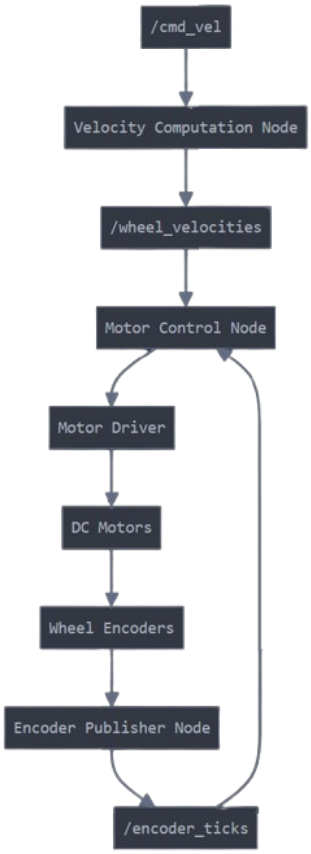


Figure 3.5: Closed-Loop Motor Control System Architecture

Figure 3.5 illustrates the architecture of the closed-loop motor control system. As shown in the diagram, the system incorporates feedback from wheel encoders to adjust motor outputs dynamically, ensuring precise control over the robot's movement.

The core of the control system is built around a control loop that runs at a fixed frequency. This loop constantly monitors the robot's state and makes necessary adjustments to maintain the desired movement. The controller uses two main strategies for control: course correction and speed control.

For speed control, the controller calculates the average velocity of the two wheels combined together. Instead of converting it to a velocity it is kept as a tick rate which is the amount

of encoder ticks per second, where a tick is rising edge pulse received from the rotary encoders. The speed error is the current average tick rate subtracted from the desired tick rate. The controller operates in a discrete-time system, updating at a fixed frequency. At each time step, it calculates the tick rates based on the change in encoder ticks:

$$\text{tick\_rate} = \frac{\text{current\_ticks} - \text{previous\_ticks}}{\Delta t} \tag{3.9}$$

where Δt is the time interval between control loop iterations.

The speed correction is then calculated using proportional control:

$$\text{speed correction} = K_s \cdot \text{speed error} \tag{3.10}$$

where K_s is the speed correction factor.

For course correction, the controller computes the difference in encoder ticks between the left and right wheels, adjusted for any initial offset. The course correction is then calculated:

$$\text{course\_correction} = K_c \cdot \text{adjusted\_tick\_difference} \tag{3.11}$$

where K_c is the course correction factor.

These corrections are added to a base PWM value. To ensure the PWM values remain within valid ranges, they are clamped so that they cannot exceed the PWM range of the pins.

When transitioning between different movement states the system incorporates a state detection mechanism. When the robot begins moving in a straight line the controller resets its internal tick difference offset. This eliminates any accumulated errors and ensures that the robot starts its straight-line motion from a clean slate which ensures the movement is precise.

Proportional control for both the course correction and speed control is used. The response of the course correction and speed correction can be tuned by editing gain variables. This allows for easy adjustment of the robot's behaviour.

This closed-loop design allows the robot to adapt to its environment and can overcome input disturbances from external factors. The combination of course correction and speed control allows the robot to move reliably in a straight line, which is crucial for many robotic applications.

3.5 Design Considerations

The system design is optimised for real-time performance, scalability and modularity. These include utilizing ROS 2's Data Distribution Service for efficient inter-node communication as well as implementing a multi-threaded approach in the SLAM node.

Chapter 4

Implementation

4.1 Hardware Setup

4.1.1 Robotic Platform Assembly

The SLAM system was implemented on an existing robotic platform which integrates various hardware components to create a functional and efficient mobile platform. Figure 4.1 provides an overview of the assembled robot, highlighting key components and their arrangement.

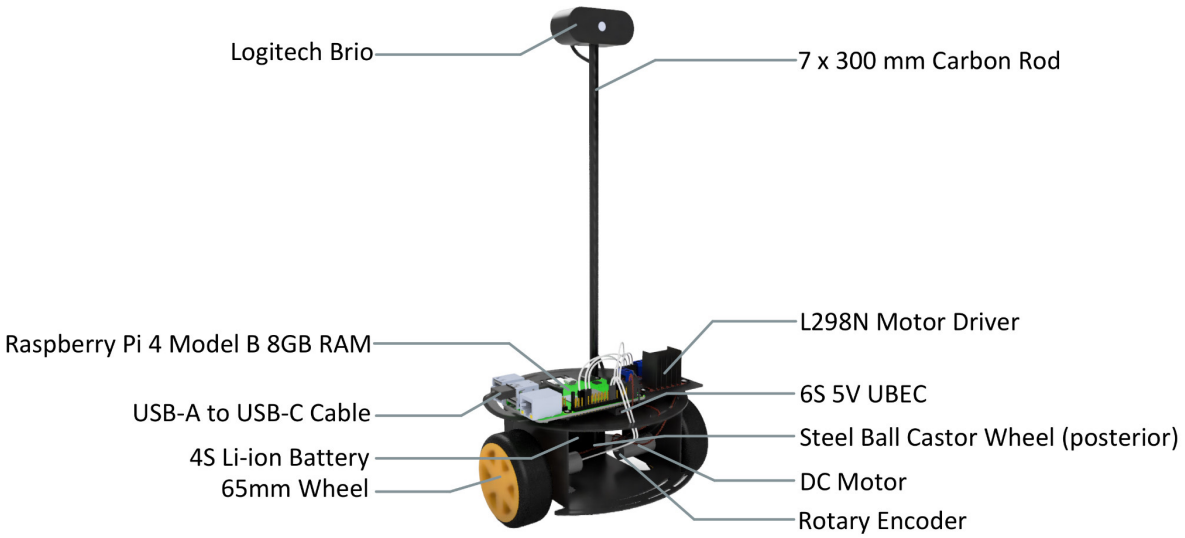


Figure 4.1: Assembled SLAM Robot with Key Components.

Figure 4.1 shows the complete assembly of the robotic platform including key components such as the Raspberry Pi 4 Model B, L298N Motor Driver, DC motors, rotary encoders and the Logitech Brio webcam mounted on a carbon rod for an elevated perspective.

The robotic platform chosen for this project was the differential drive DFROBOT Turtle 2WD. This platform is a stable base for the SLAM system due to its simplicity in both mechanical design and control. The assembly process involved several key components:

1. Chassis: The DFROBOT Turtle 2WD chassis served as the foundation for the robot. This pre-fabricated chassis is designed to accommodate the necessary components while providing a balanced weight distribution.
2. Drive System: Two DC motors were mounted on the chassis, one for each drive wheel. These motors provide independent control of each wheel which allows the differential drive mechanism to control the robot.
3. Motor Control: To manage the DC motors, a Motor Driver Dual H-Bridge Module L298N was installed. This module allows for bidirectional control of both motors, enabling forward, reverse, and turning movements of the robot with varying levels of speed.
4. Power Distribution: A 5V UBEC (Universal Battery Elimination Circuit) was incorporated into the design to provide a stable power supply for the Raspberry Pi. This ensures that the main computational unit receives a consistent 5V power source, which is critical for reliable operation.
5. Power Source: A 4S Li-ion battery pack was installed to power the entire system. This battery provides the necessary voltage and capacity to run both the motors and the electronic components for extended periods.

The wiring of these components followed the wiring diagram in Figure B.2 in Appendix B.4:

- The DC motors were connected to the output terminals of the L298N motor driver.
- The L298N motor driver was wired to the Raspberry Pi's GPIO pins for control signals.
- The 5V UBEC was connected to the battery pack and provided power to the Raspberry Pi.
- The L298N motor driver was directly powered by the battery pack to handle the higher current requirements of the motors.

Wire management and secure connections were one of the main foci as this would ensure reliability and prevent any short circuits or loose connections during operation. Components were compact and as close to the ground as possible to lower the centre of gravity.

4.1.2 Sensor Integration

The two types of sensors used in the system are a USB webcam and two rotary encoders. A Logitech BRIO webcam was selected due to its high-quality imaging capabilities and USB connectivity. The camera was directly connected to the Raspberry Pi 4B's USB ports. Two rotary encoders were integrated into the system to supply odometry data to the system. These encoders have 10 teeth per revolution which generates 10 logic HIGH signals for each wheel rotation. The two encoders were connected to the Raspberry Pi via the GPIO pins. Each encoder has three connections: power (5V), ground and signal. The signal wire from each encoder was connected to a designated GPIO pin on the Raspberry Pi which allows the Pi to read the sensor data directly.

4.1.3 Computational Hardware Configuration

A Raspberry Pi 4 Model B was chosen for the onboard compute unit because of its 8GB of RAM and fast 1.8GHz processor. It was chosen for its compact size and powerful processing capabilities. The Raspberry Pi runs Ubuntu Server 22.04 which is a powerful operating system (OS) and is compatible with ROS 2. The L298N motor driver has a 5V regulator however it has a maximum output current of 500mAh which is far below the 3A that the Raspberry Pi requires so a 5V UBEC connects the 4S Li-ion battery to the Raspberry Pi which provides the stable power source for the compute unit. The Raspberry Pi's USB ports are utilized for connecting the Logitech BRIO webcam and the GPIO pins interface with the rotary encoders and the L298N motor driver. The Raspberry Pi's wireless connectivity is essential to how the system operates as instructions and data can be sent over a local network. This compact yet powerful setup provides the necessary computational resources for running complex SLAM algorithms while maintaining the robot's mobility and energy efficiency.

4.2 ROS 2 Environment Setup

4.2.1 ROS 2 Installation and Configuration

The ROS 2 Humble distribution was installed on both the Raspberry Pi 4 and the base station computer. Due to both systems running the same version of Ubuntu and ROS 2, this creates a distributed computing environment for the SLAM system. This dual-installation approach provides flexibility in processing capabilities and allows for the potential offloading of computationally intensive SLAM operations to the more powerful base station if needed.

The ROS 2 Humble detailed installation is in Appendix B.2. The installation included core ROS 2 packages and development tools necessary for robot control as well as sensor

data processing. After ROS 2 has been installed the environment was configured to automatically source the setup file in the .bashrc which ensures that ROS 2 commands are available in every new terminal session and do not need to be sourced each time a new terminal is started.

To facilitate communication between the Raspberry Pi and the base station, ROS 2's native Data Distribution Service (DDS) was configured. This middleware allows for efficient, real-time data exchange over the network, enabling topics such as sensor data, control commands, and SLAM outputs to be shared seamlessly between the robot and the base station.

The complete installation and configuration process, including specific commands and troubleshooting steps, is documented in the project's GitHub repository, accessible via the link provided in Appendix B.1. This documentation ensures reproducibility and serves as a reference for future modifications or system upgrades.

By leveraging ROS 2's distributed architecture, this configuration provides a scalable and flexible foundation for the SLAM system, allowing for adaptive processing allocation based on computational demands and available resources.

4.2.2 Creating the ROS 2 Workspace

There are two ROS 2 workspaces that were created. The first was on the Raspberry Pi and the second was on the base station. The Pi contains the camera node, encoder node, velocity computation node and motor control node. These nodes are responsible for the closed-loop control of the robot, to ensure it travels in a straight line when it is being commanded to do so, as well as the streaming of the camera feed with each frame's associated timestamp to the base station for the offloading of the SLAM computations.

On the Pi, a python ROS 2 package was created and the python nodes were created in the correct directory as detailed in Figure B.3. Once the nodes have been made the setup.py and package.xml files were edited to link the python nodes to the package structure and to include the dependencies for the nodes. A launch file was created so that the execution of just one file is required to start all the nodes required for the system to start the SLAM process. Lastly, the package is built and sourced.

The base station has a combination of C++ and Python nodes. The python node required for the system to operate is the robot remote control node which sends movement instructions to the robot via the /cmd_vel topic. An additional directory is added to the project files that contain the __init__.py and robot_remote_control.py files. The python node is made an executable and the project CMakeLists.txt file and package.xml files were edited to include the python node functionality and to link to the node as seen in Figure B.3. The launch file directory and file were created and the nodes were included in the launch file with the respective launch arguments. The package was built and sourced to conclude the ROS 2 workspace setups on both platforms.

4.2.3 Package Structure and Organization

There are two ROS 2 packages being used in the system as seen in Figure 4.2. The ros2_robot_package is the package on the Raspberry Pi and contains the nodes responsible for the closed-loop motor control as well as the publishing of the camera frames with their associated timestamps to the base station. The second package is the orbslam3 package which is on the base station and is responsible for subscribing to the camera feed as well as controlling the robot remotely. In the following section, the details of the nodes and the topics used will be expanded on.

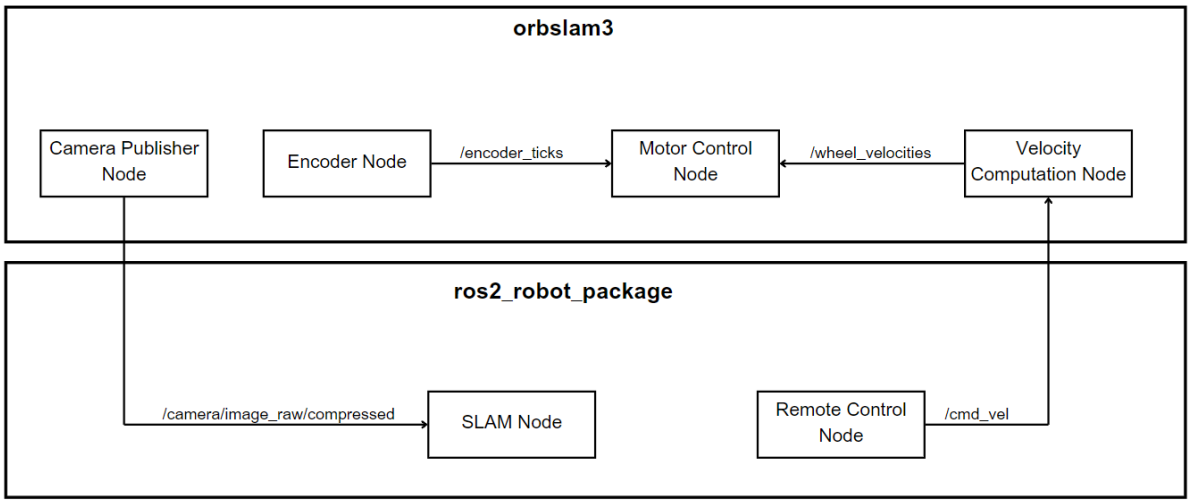


Figure 4.2: System packages, nodes and topics.

4.3 Node Implementation

4.3.1 Camera Node

The camera node is crucial for operating the SLAM system as it is responsible for capturing and broadcasting camera frames from the Logitech BRIO webcam. The nodes were implemented using Python, ROS 2 and OpenCV (detailed in Appendix C.1). The node publishes compressed image messages to the `/camera/image_raw/compressed` topic at 30 Hz.

The CompressedCameraPublisher class initializes the webcam with a 640x360 pixel resolution, balancing image quality and computational efficiency. During each iteration, the node captures a frame, compresses it to JPEG format, and encodes it into a ROS 2 CompressedImage message. Crucially, the node embeds the current ROS 2 clock time into each message's header. This timestamp enables accurate synchronization of the current frame and can be used to calculate the time difference between images for use with the SLAM framework. Compressed images optimize bandwidth usage, facilitating efficient data transmission across the ROS 2 network. By encapsulating camera operations within

a modular structure, the camera node exemplifies ROS 2's design principles, enhancing system flexibility and simplifying integration with various SLAM implementations or additional image processing components.

4.3.2 Encoder Node

The encoder node is responsible for odometry in the SLAM system. The node interfaces with the rotary encoders attached to the robot's wheels. It is implemented in Python using ROS 2 and the RPi.GPIO library (detailed in Appendix C.2). It publishes encoder tick counts to the /encoder_ticks topic at a high frequency of 1000 Hz. The EncoderPublisher class initializes the GPIO pins for both the left and the right encoders as well as setting up pull-up resistors to ensure stable readings. To reduce false readings the node implements a debouncing mechanism. This mechanism has a threshold of 5 consecutive consistent readings before registering a state change. If a state change is detected the tick value for the associated wheel is incremented. The published topic message is an array of two integers representing the cumulative tick counts for both the left and right wheels.

4.3.3 Velocity Computation and Motor Control Nodes

The VelocityComputationNode in Appendix C.3 converts robot movement commands into individual wheel velocities. The node subscribes to the /cmd_vel topic which receives Twist messages containing normalized linear (x) and angular (z) velocity commands. When a command is received the node scales the normalized input using predetermined speeds (0.25 m/s linear, 2.5 rad/s angular). The node currently is set to a target speed but with minor adjustments, it can be set to purely calculate the wheel speeds for any input linear and angular velocities which can help facilitate navigation in future implementations. It then uses differential drive kinematics to compute the required wheel velocities based on the robot's wheelbase and wheel radius. The node then publishes the individual wheel speeds to the /wheel_velocities topic using the Float32MultiArray message type.

The SimpleMotorControlNode in Appendix C.4 controls the motors using the Raspberry Pi's GPIO pins. It subscribes to two topics: /wheel_velocities for receiving desired wheel speeds and the /encoder_ticks for feedback from wheel encoders. The node implements a closed-loop control system running at 10 Hz that combines speed and course correction for straight-line motion. It uses encoder feedback to maintain both consistent speed and straight-line trajectory. The control system applies the needed corrections through PWM signals running at 100Hz. The gain values for the speed and course correction can be edited for desirable response characteristics by adjusting the values of speed_correction_factor (0.1) and course_correction_factor (0.8). The node manages four GPIO pins for motor direction control and two PWM-capable pins for speed control. For an additional layer of safety and to ensure the controller operates correctly the output PWM values are clamped between 0 and 100%.

4.3.4 ORB-SLAM3 Node

This ROS 2 implementation of ORB-SLAM3 consists of a monocular SLAM node (Appendix C.6) that processes compressed camera images for real-time visual SLAM. The system is initialized in mono.cpp (Appendix C.6.2), which loads an ORB vocabulary (used for place recognition and loop closure) and creates the SLAM system with specified configuration files that are inputted as launch arguments. The MonocularSlamNode class (Appendix C.6.3)subscribes to compressed images on the topic /camera/image_raw/compressed and processes them in the GrabImage callback function. When an image arrives, it's decompressed, converted to grayscale, and resized to 640x360 pixels if necessary. The node maintains a normalized timestamp system starting from 0.0 for the first frame and calculates subsequent timestamps as the time difference from the initial frame. Each processed image is then passed to the ORB-SLAM3 system's TrackMonocular function along with its timestamp for simultaneous localization and mapping. When the node is destroyed, it properly shuts down the SLAM system and saves both the camera trajectory in TUM format and the generated point cloud to disk. During SLAM operation the system has a GUI interface that shows the point cloud, estimated trajectory and the current frame with ORB features overlayed with easy user controls.

4.3.5 Remote Control Node

To control the robot a remote control node was implemented on the base station (see Appendix C.6.1). This node is responsible for controlling the movement of the robot and it does this by sending /cmd_vel instructions to the robot using the built-in topic publishing and subscribing functionality of ROS 2. The user can press the keyboard keys w, a, s, d and x to move the robot forward, rotate anti-clockwise, move backward, rotate clockwise and stop the robot respectively. When the node recognises that there has been a keyboard press it changes state to the desired new state. The message structure sent to the robot over the /cmd_vel topic is of type geometry_msgs/Twist and it has a linear velocity followed by an angular velocity. The node assigns unit velocity values for the corresponding keyboard press. The motor control node then decodes this message to use it with the motor control.

4.4 SLAM System Integration

4.4.1 ORB-SLAM3 Integration

The integration of ORB-SLAM3 presented several challenges due to its age and lack of built-in ROS 2 support. Appendix B.2 details the installation process, however it deviated significantly from the standard GitHub repository instructions. Due to the algorithm being more than 3 years old, it required modifications to run on modern library versions

and two separate platforms. The build files needed adjustments to ensure compatibility and to ensure it maintains the core functionality.

A limitation of the original ORB-SLAM3 implementation was there were no executables designed for webcam usage. Custom executables were developed to interface with the Logitech BRIO webcam. These executables were first created and tested independently to ensure proper functionality with the camera hardware to ensure the webcam can be used for live SLAM operations. Once it had been successfully implemented using C++ executables a ROS 2 C++ node was implemented to interface with the ORB-SLAM3 framework. This allowed for efficient real-time SLAM to take place.

4.4.2 LDSO Integration

The integration of LDSO into the SLAM system presented its own set of challenges some of which were similar to those experienced by the ORB-SLAM3 implementation. As outlined in Appendix B.2, the installation process for LDSO deviated from the standard procedure due to dependency conflicts with modern system libraries. CMake files and source code required modifications to successfully build on the base station and the Raspberry Pi.

4.4.3 Camera Calibration and Selection

The accuracy of visual SLAM systems heavily relies on precise camera calibration. Initially, a CSI (Camera Serial Interface) camera was attempted to be used (wide-angle Raspberry Pi Camera). However significant compatibility issues were encountered when trying to implement it on the Raspberry Pi due to the Ubuntu Server OS.

The reason for this incompatibility is because of the differences between Ubuntu and Raspberry Pi OS. Ubuntu lacks the drivers and firmware necessary for compatibility. Raspberry Pi OS includes the requirements to support the CSI camera interface. The CSI camera relies on Broadcom binary blobs and MMAL (Multi-Media Abstraction Layer) libraries. Ubuntu Server 22.04 is a 64-bit OS and the CSI camera is designed for use on a 32-bit OS.

Due to this a USB webcam was used (Logitech BRIO) which offers better compatibility across different operating systems. To calibrate this camera a custom Python script was developed and utilized (see Appendix C.5 for the full script). This script implements a chessboard-based calibration method using the OpenCV library.

The calibration process begins with capturing camera frames. The script sets the resolution to 640x360 pixels and captures a series of 15 images of a 7x9 chessboard pattern from various angles and distances. Each captured image is then processed to detect the chessboard corners with sub-pixel accuracy using OpenCV's findChessboardCorners and cornerSubPix functions.

Once the corners are detected they are used as inputs to OpenCV's calibrateCamera function. This function implements Zhang's method for camera calibration. This calculates the camera's intrinsic matrix and distortion coefficients. The parameters are further refined using cv2.getOptimalNewCameraMatrix to minimize reprojection errors. The script outputs the calibrated parameters, including focal lengths, principal point coordinates, and distortion coefficients, in a format directly compatible with ORB-SLAM3's configuration files.

The camera calibration ensures that the SLAM system can accurately interpret the visual data from the webcam. By using a custom calibration script the specific camera model can be calibrated and any other camera models in the future. The switch to a USB webcam ultimately provided a more flexible and widely supported solution for our Ubuntu-based robotic platform.

Chapter 5

Results and Discussion

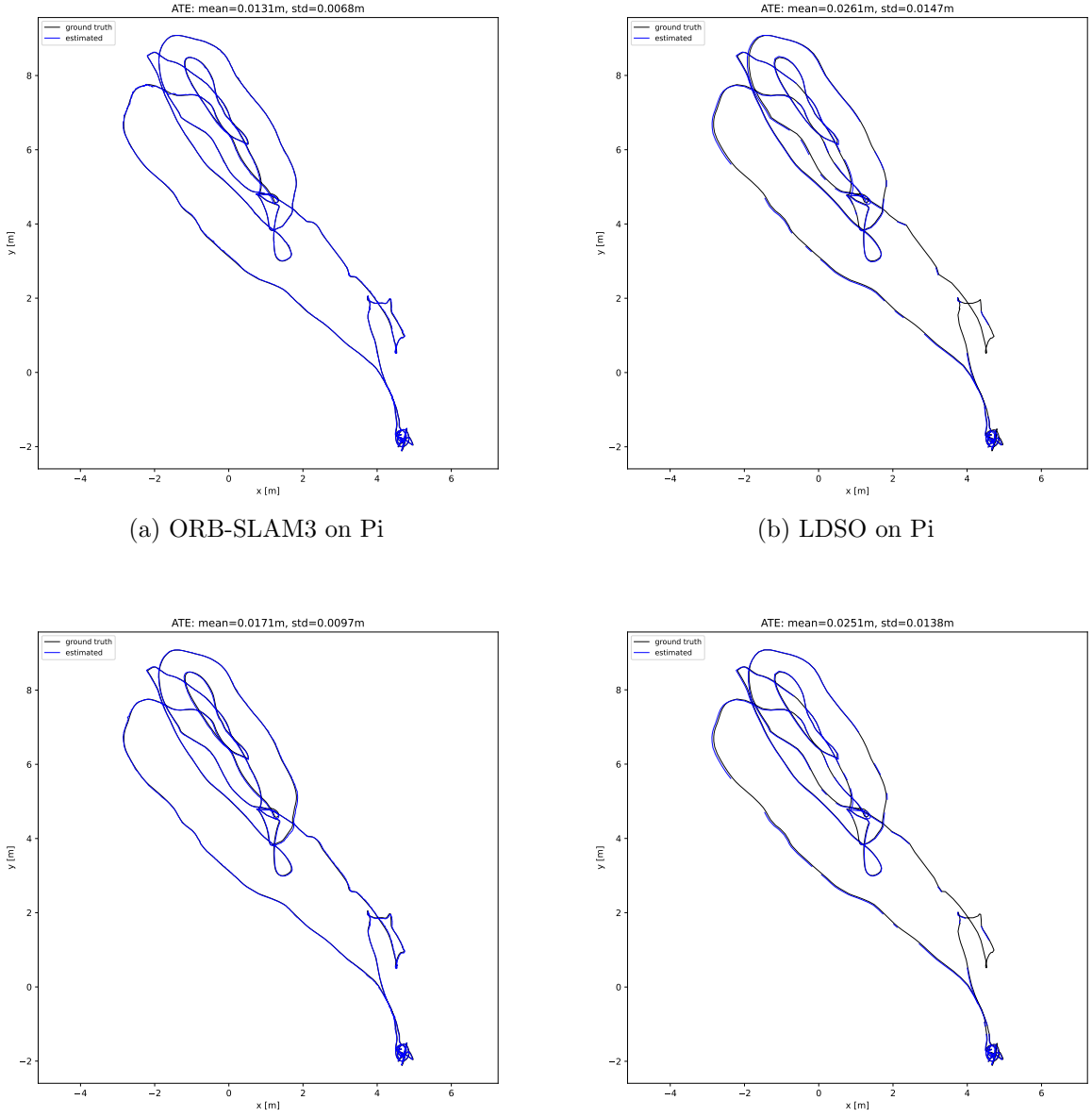
The SLAM system was evaluated using simulations and real-world experiments to test its performance. The following sections are divided into the simulation results and the experimental results. The simulation studies allowed for controlled testing of the system's core algorithms and the experimental results demonstrate the system's performance in real-world scenarios. These assessments provide a comprehensive validation of the SLAM system's capabilities and its readiness for practical applications.

5.1 Simulation Results

ORB-SLAM3 and LDSO were evaluated on different hardware platforms. The first evaluation was a comprehensive comparison between platform-algorithm combinations on the EuRoC dataset. This dataset was recorded from a Micro Aerial Vehicle (MAV) and provides images at 20 fps. Accurate ground truth trajectories are also included making it ideal for benchmarking visual SLAM systems. The datasets used were the Machine Hall (MH) and the Vicon Room (V1). The two datasets are different types of environments, the MH is a large indoor environment and the V1 is a small room. Each of the two datasets has three levels of difficulty that are designed to gradually push the SLAM system to its limits.

Figure 5.1 presents a comparison of ORB-SLAM3 and LDSO trajectories against the ground truth for the MH01 dataset. The dataset was run on both a base station as well as a Raspberry Pi. The plots demonstrate the general capability of both algorithms to track the MAV's path. The trajectory shows minor deviations which are particularly visible in the more complex sections of the trajectory due to rapid translation and rotation occurring. This initial comparison provides a brief overview of the systems' accuracies.

The Absolute Trajectory Error (ATE) was calculated for each algorithm-platform combination on datasets MH01, MH03, MH04, V101, V102, and V103. Figure 5.2(a) illustrates the mean errors for these experiments, while Figure 5.2(b) presents the corresponding standard



(a) ORB-SLAM3 on Pi

(b) LDSO on Pi

(c) ORB-SLAM3 on PC

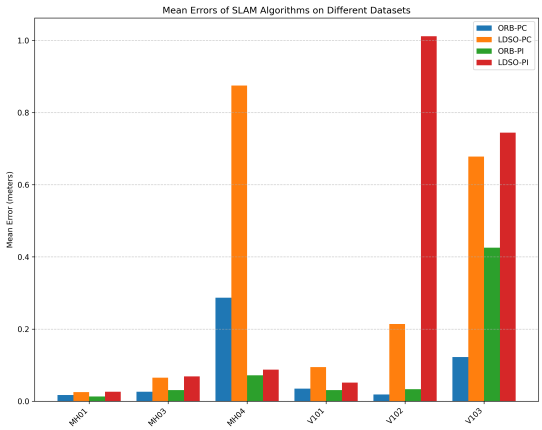
(d) LDSO on PC

Figure 5.1: Comparison of ORB-SLAM3 and LDSO with different platforms on the MH01 dataset

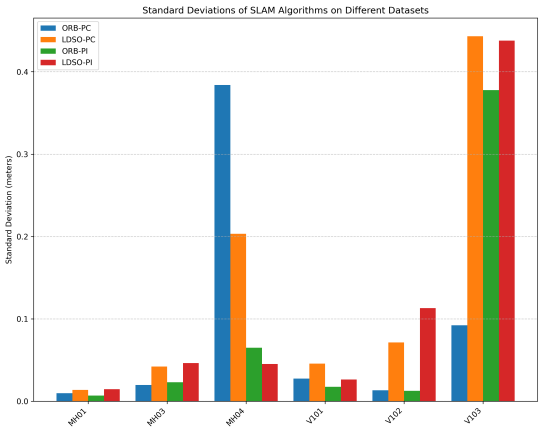
deviations. These graphs offer a comprehensive view of the algorithms' performance across various environments and difficulty levels.

Figures 5.2(a) and 5.2(b) compare the mean and standard deviations of the ATE results between the platforms, algorithms and datasets. The graphs indicate that ORB-SLAM3 generally outperforms LDSO across most datasets and exhibits lower mean errors and standard deviations. The performance gap is particularly evident in the datasets that are more complicated like the MH04 and V103 sequences. The graphs also highlight a difference in performance between the PC and Raspberry Pi implementations which is particularly noticeable for ORB-SLAM3. In the easy-difficulty datasets, the PC version of ORB-SLAM3 and the Pi version have similar performances even though the PC has

more processing power. The PC has a 6-core (6-thread) 4.2 GHz Ryzen 5 500 compared to the 4-core 1.8 GHz Broadcom BCM2711 ARM processor on the Raspberry Pi. The reason why both systems have similar simulation results in some datasets is because the datasets are not being run in real time and the SLAM system waits for the current frame to be processed before continuing to the next frame. This allows systems with weaker processing power to have good simulation outputs. As the complexity of the environments increases from MH01 to MH04 and V101 to V103 both algorithms show increasing error rates. This trend is expected, as these sequences introduce more dynamic elements and challenging lighting conditions.



(a) Comparison of the mean error of different SLAM algorithms and platforms

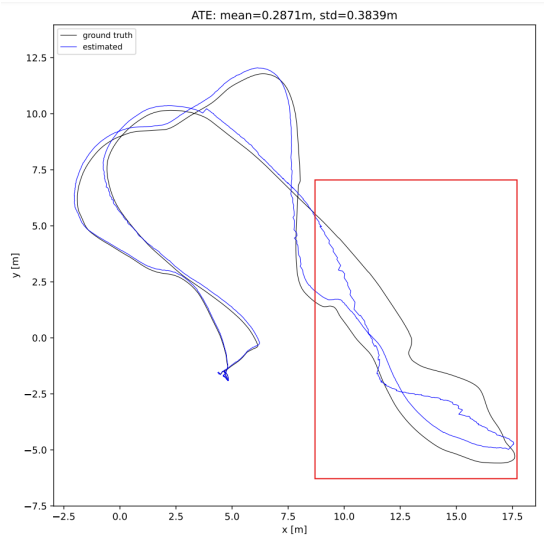


(b) Comparison of the standard deviation of the error of different SLAM algorithms and platforms

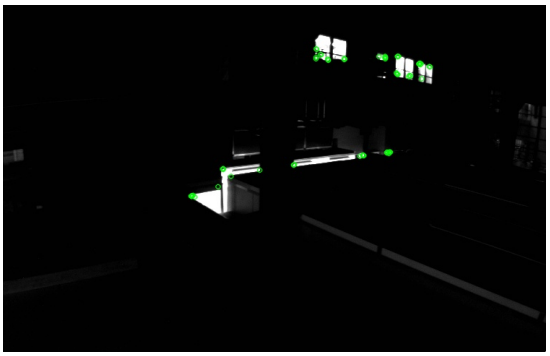
Figure 5.2: Comparison of ORB-SLAM3 and LDSO on different datasets and platforms

Figure 5.3(b) showcases a frame from the MH04 dataset where low-light conditions posed significant challenges to ORB-SLAM3. In this frame, only 67 ORBs were detected as seen by the green circles in Figure 5.3(b), this is far below the number of ORBs that would allow ORB-SLAM3 to accurately track the environment and explains the degraded tracking accuracy for the algorithm on the MH04 dataset when the MAV encounters this low-light environment. Figure 5.3(a) highlights the area of the trajectory (marked in red) where there were low-light conditions present. Both algorithms, but particularly ORB-SLAM3, struggled to maintain accurate tracking during the low-light conditions.

The performance issues in low-light scenarios, as evidenced in Figures 5.3 and 5.3(a), are particularly noteworthy for monocular SLAM systems. Both ORB-SLAM3 and LDSO rely heavily on visual features for mapping and localization and in low-light conditions there is little camera data that is useful for such SLAM systems. ORB-SLAM3 is an indirect technique as ORBs need to first be detected for tracking to occur, whereas LDSO uses direct techniques which rely on a sufficient intensity gradient being present in the frame for it to track the environment. In dark environments, the reduced number of detectable features can lead to tracking failures or increased drift in pose estimation for ORB-SLAM3. LDSO performs better than ORB-SLAM3 in low-light conditions and tracks the environment well. LDSO is less accurate than ORB-SLAM3 in the simulations as seen by its inferior mean and standard deviation trajectory errors in Figure 5.2(a) and Figure 5.2(b).



(a) The estimated and ground truth trajectories for ORB-SLAM3 on the PC running the MH04 dataset



(b) A frame from the MH04 dataset corresponding to an unlit section of the trajectory

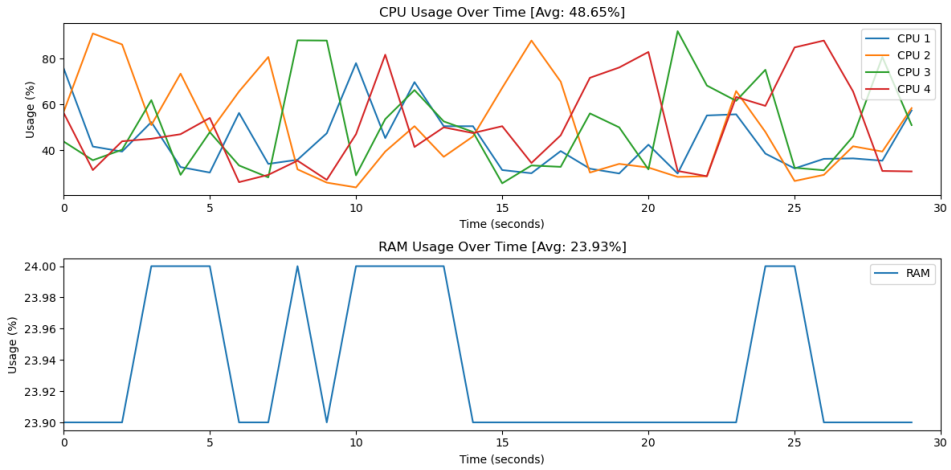
Figure 5.3: A frame from the MH04 dataset and the area of the MH04 dataset that was dark

In conclusion, these simulation results provide valuable insights into the behaviour of ORB-SLAM3 and LDSO across different environments and hardware platforms. They underscore the need for robust feature detection and matching algorithms, especially in challenging lighting conditions, and highlight the ongoing challenges in monocular SLAM.

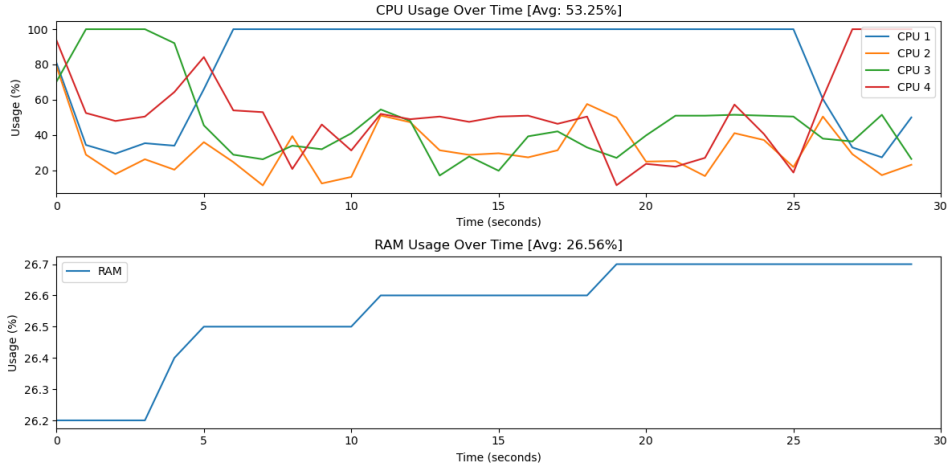
5.2 Experimental Testing

5.2.1 Processing Power Analysis

ORB-SLAM3 and LDSO were evaluated on both a PC and a Raspberry Pi platform to assess their computational demands. Tests were conducted to compare the platform-algorithm combination resource requirements. To do so the camera resolution was set to 640 by 480 pixels and the camera was positioned in a well-lit environment.



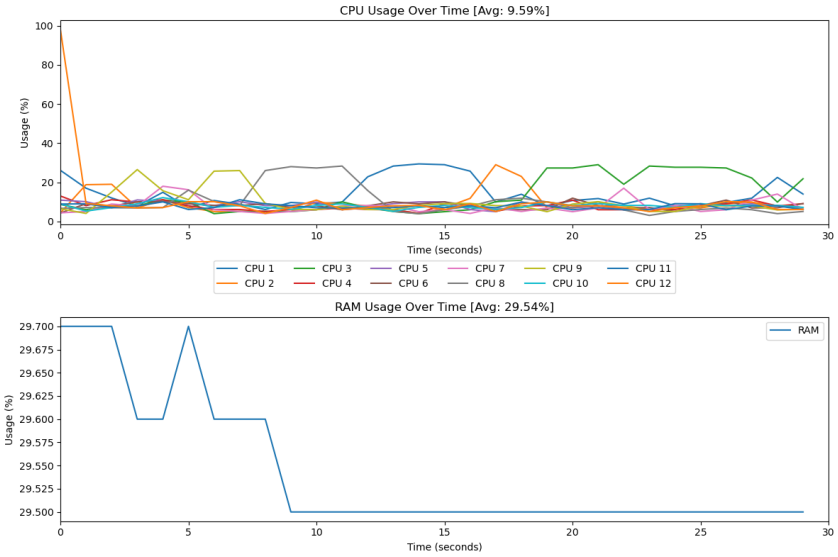
(a) CPU and RAM usage on Pi running ORB-SLAM3



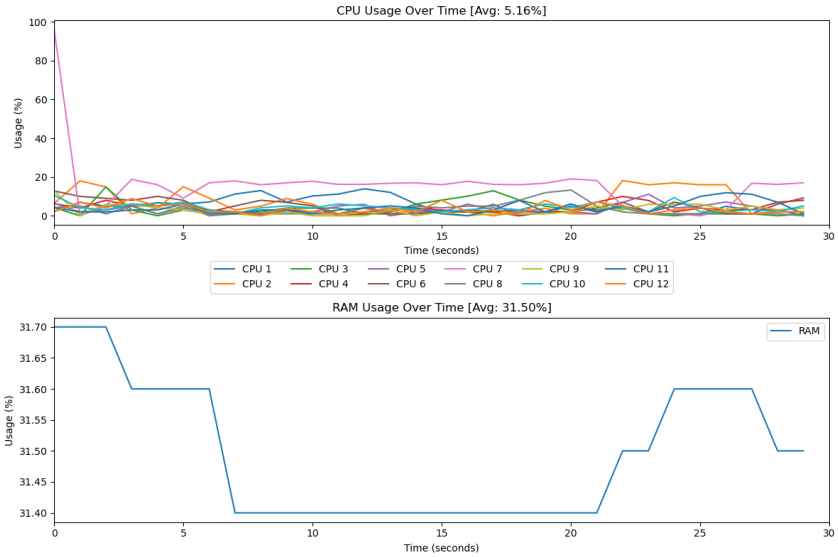
(b) CPU and RAM usage on Pi running LDSO

Figure 5.4: Comparison between LDSO and ORB-SLAM3 processing power usage on a Pi

Figure 5.4(a) presents ORB-SLAM3's performance on the Raspberry Pi. Despite the platform's limited resources, the algorithm maintains an average CPU usage of 48.65% across its 4 cores. RAM utilization averages 23.93% of the available 8 GB, exhibiting minimal variations over the period. Figure 5.4(b) shows LDSO's resource consumption on the Raspberry Pi. The algorithm's CPU usage averages 53.25%, with substantial fluctuations throughout the test and cores reaching 100% usage throughout the 30-second period. RAM usage remains relatively constant at an average of 26.56%, indicating efficient memory management on the constrained hardware.



(a) CPU and RAM usage on PC running ORB-SLAM3



(b) CPU and RAM usage on PC running LDSO

Figure 5.5: Comparison between LDSO and ORB-SLAM3 processing power usage on a PC

Figure 5.5(a) shows the resource utilization of ORB-SLAM3 on the PC. The CPU usage on all 12 cores shows an average of 9.59% with some sporadic spikes reaching up to 20% on individual cores. RAM usage remains relatively stable with an average of 29.54% of the available 16 GB being used. The RAM usage has minor fluctuations throughout the 30-second test period. Figure 5.5(b) shows the performance of LDSO on the PC and the algorithm demonstrates lower CPU utilization with an average of 5.16% and the RAM consumption averages 31.50%.

These results highlight the significant difference in computational demands between ORB-SLAM3 and LDSO. LDSO demonstrates remarkable efficiency due to its direct tracking technique and has far less CPU overhead. Conversely, ORB-SLAM3 tends to have higher

CPU usage, potentially limiting its applicability in power-constrained environments.

5.2.2 SLAM Accuracy Analysis

This section is a comparative analysis of the accuracy achieved by ORB-SLAM3 and LDSO algorithms when implemented on both a Raspberry Pi and a PC. To evaluate the SLAM systems data was collected from driving the robot on a predefined loop and then comparing the performance of the platform-algorithm combinations with the ground truth.

Camera Comparison

The choice of camera and its positioning are crucial factors in the performance of monocular visual SLAM systems as the camera is the only sensor being used to beform the SLAM. This study compared the Logitech Brio and Logitech C525 webcams and ultimately selected the Brio for its superior capabilities. Figure 5.6(b) shows the Brio's superior low-light performance and larger field of view (FOV). The C525's FOV is superimposed with green vertical lines in Figure 5.6(b). Figure 5.6(a) shows the dark FOV of the C525. The attributes mentioned are particularly advantageous for SLAM applications as they allow the robot to SLAM in challenging lighting conditions and confined spaces. The wider FOV allows for capturing more environmental features, critical for robust feature tracking and mapping.

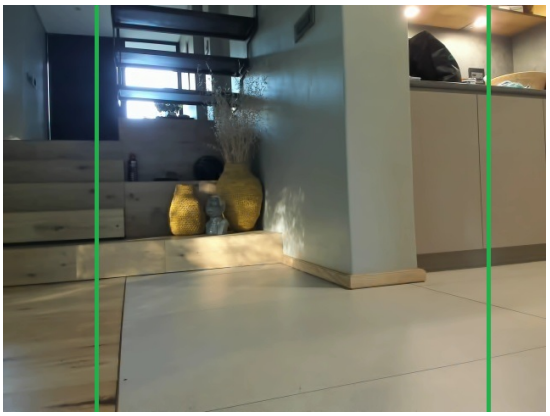
Camera height significantly impacts feature detection which is crucial for ORB-SLAM3's performance. Figure 5.7 shows the ORB features detected at two different camera heights. The camera position 370 mm above the ground yielded 587 ORB features, while the one with a height of 80 mm detected only 429 features. This is a 37% increase in detected features at the higher position. The reason for this large increase in detected ORB features is at a higher position teh camera has a better visual context of the environment and a higher vantage point leads to the camera having broader scene. At a higher position there are fewer visual occlusions due to it being able to minimise nearby objects. With these findings it was decided that the Logitech Brio at a height of 370 mm would be used for benchmarking the SLAM systems in the following sections.

SLAM System Testing

To assess the accuracy of both SLAM algorithms, a controlled experimental setup was used:

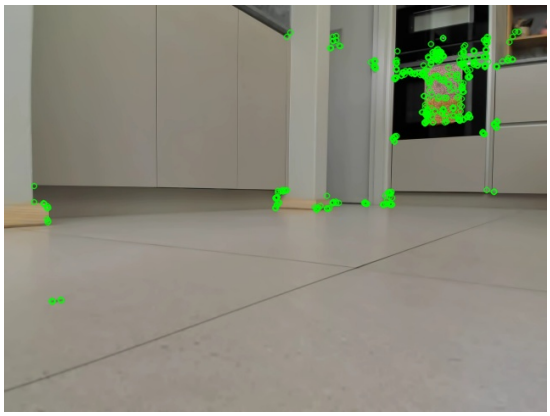
1. A predefined loop path was established in an indoor environment to serve as ground truth seen in Figure 5.8.

(a) Logitech C525 webcam frame in low-light conditions

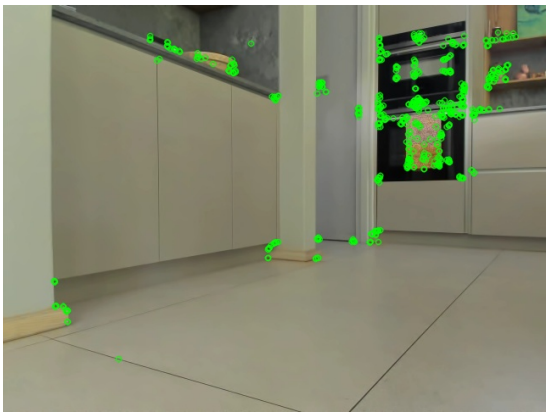


(b) Logitech Brio webcam frame in low-light conditions indicating the field of view of the C525

Figure 5.6: A comparison between the Loitech C525 and Logitech Brio webcams in low-light conditions



(a) Logitech Brio webcam at a height of 80 mm above the ground with detected ORBs overlayed



(b) Logitech Brio webcam at a height of 370 mm above the ground with detected ORBs overlayed

Figure 5.7: A comparison between having the Logitech Brio webcam at a height of 80mm versus 370mm above the ground

2. The mobile platform either was running SLAM locally on the Raspberry Pi or the SLAM was being offloaded to the base station.
3. Both ORB-SLAM3 and LDSO were run on each hardware platform, processing real-time video input from a calibrated camera.
4. The estimated trajectories from each algorithm were recorded and compared against the ground truth path.

Results and Analysis

The comparison of LDSO performance on the Raspberry Pi (Figure 5.9(a)) and the PC (Figure 5.9(b)) reveals stark contrasts in trajectory estimation accuracy and consistency.

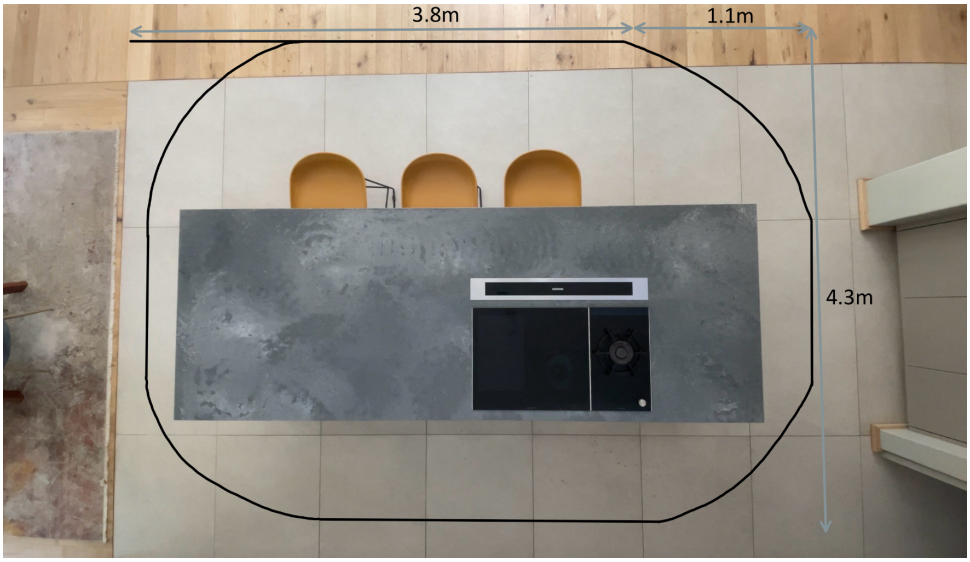
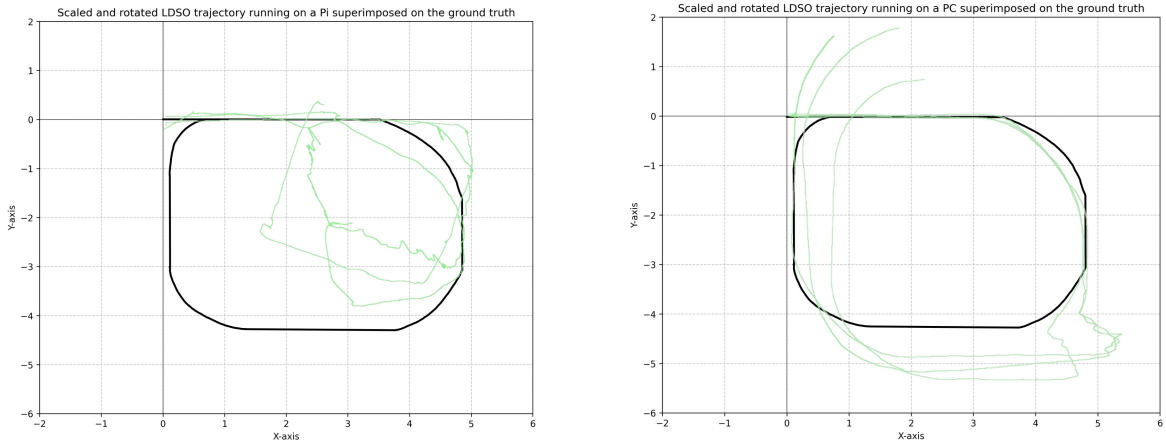


Figure 5.8: Robot trajectory ground truth.



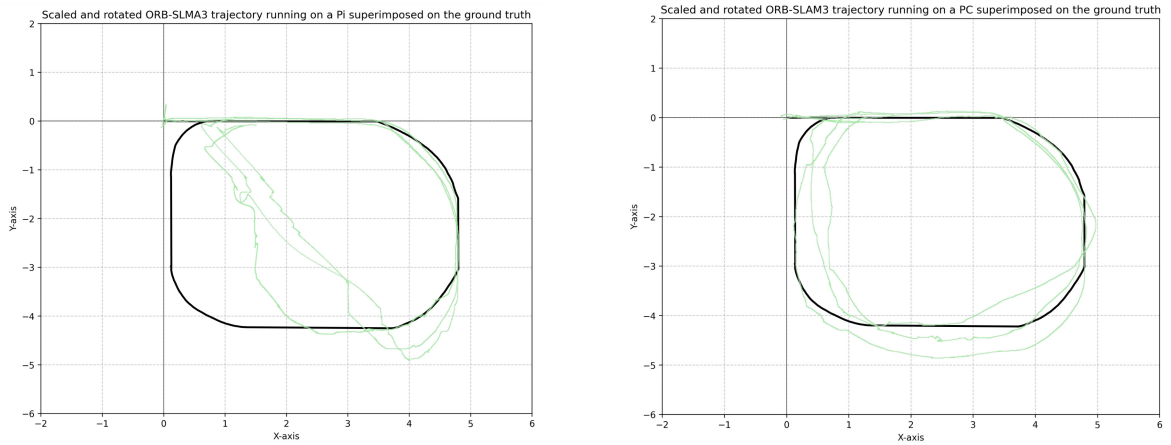
(a) LDSO vs. Ground Truth on Pi

(b) LDSO vs. Ground Truth on PC

Figure 5.9: A comparison between LDSO running on the Pi vs. on the PC

Generally, the PC version of LDSO has better accuracy than the Pi version as seen by the better adherence to the ground truth trajectory. The PC version also had effective loop closure whereas the Pi exhibits deviations and an inaccurate trajectory. The Pi shows poor loop closure capabilities, significant drift and inaccurate scale estimation. The major difference between the versions is observed during cornering where the Pi has an erratic trajectory. The PC's superior performance is characterized by better tracking and more robust real-time processing capabilities. The Pi's limited computational resources contribute to its degraded performance which affects its ability to maintain accurate trajectory estimation in real-time. This comparison demonstrates the influence that computational hardware has on SLAM algorithm performance. The PC outperforms the Raspberry Pi in maintaining accuracy and robustness to environmental challenges when running LDSO.

ORB-SLAM3 trajectories on the Raspberry Pi (Figure 5.10(a)) and PC (Figure 5.10(b)) show substantial differences compared to the ground truth. Similarly to LDSO, the PC



(a) ORB-SLAM3 vs. Ground Truth on Pi

(b) ORB-SLAM3 vs. Ground Truth on PC

Figure 5.10: A comparison between ORB-SLAM3 running on the Pi vs. on the PC

version has superior performance which is demonstrated by the closely bundled trajectories that are close to the ground truth, especially in demanding sections like the corners. The Pi implementation is less accurate than the PC version and has a larger spread and a larger mean ATE as seen in Figure 5.10(a). The pi struggled in the lower-left corner of the loop and showed a lack of scale and localisation accuracy. Both the PC and the Pi both had effective loop closure where the ending point was close to the starting location. The Pi struggles with estimating the scale in the bottom right corner which leads to degraded performance over the remaining sections of the loop.

Scale Error Comparison

The comparison of trajectory scale accuracy between ORB-SLAM3 and LDSO on the PC and the Raspberry Pi platforms reveals consistent underestimation of scale across all configurations as seen in Figure 5.11. Data was collected by aligning estimated trajectories with the ground truth by using a best-fit rotation and scaling adjustment. If this was not possible the start points were matched and the initial corners were also matched between the ground truth and the SLAM trajectory if precise alignment was not possible. The scale error is expressed as a fraction where anything between 0 and 1 means that the estimated trajectory scale is smaller than the true trajectory scale. Scale errors greater than 1 indicate the estimated trajectory scale is larger than the true trajectory scale and a scale error of 1 means that the estimated and the true trajectories are identical. LDSO on PC showed the most accurate scale estimation (mean 0.203, std dev 0.033) but exhibited significant performance degradation on the Raspberry Pi (mean 0.097, std dev 0.033). ORB-SLAM3 demonstrated more consistent performance across platforms with similar means on PC (0.094) and Pi (0.089). The standard deviation was slightly higher on the Pi (std dev 0.031 vs 0.019 on PC). The underestimation of the trajectory scale is a common challenge in monocular SLAM systems due to the lack of absolute scale information. The variability observed, particularly in LDSO on PC (range 0.160 to 0.240 as seen in Figure 5.11), highlights the sensitivity of scale estimation to environmental factors and initial conditions.

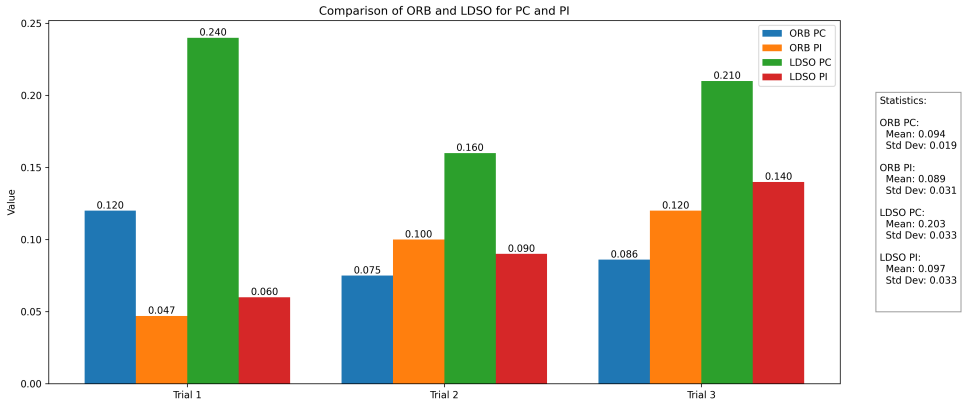


Figure 5.11: SLAM scaling error with the associated mean and standard deviations.

SLAM Processing Performance Benchmarks

Figure 5.12 reveals significant variations in processing efficiency between ORB-SLAM3 and LDSO on the PC and the Pi. On the PC LDSO demonstrates superior performance with an impressive 211.42 FPS and a low processing time per frame of 4.81 ms. ORB-SLAM3's 129.38 FPS and 7.73 ms processing time are substantially slower than that of LDSO. On the Pi, ORB-SLAM3 has an effective 7.43 FPS and 135.76 ms processing time which is sub-optimal for a monocular SLAM system as dynamic environments with fast-paced movements tend to throw off the SLAM algorithm. LDSO experiences a substantial performance drop on the Pi compared to the PC version and manages 40.61 FPS but with a significantly increased processing time of 24.71 ms. LDSO being a direct SLAM technique has less CPU overhead as it requires less processing power to operate due to there being less feature detection being done.

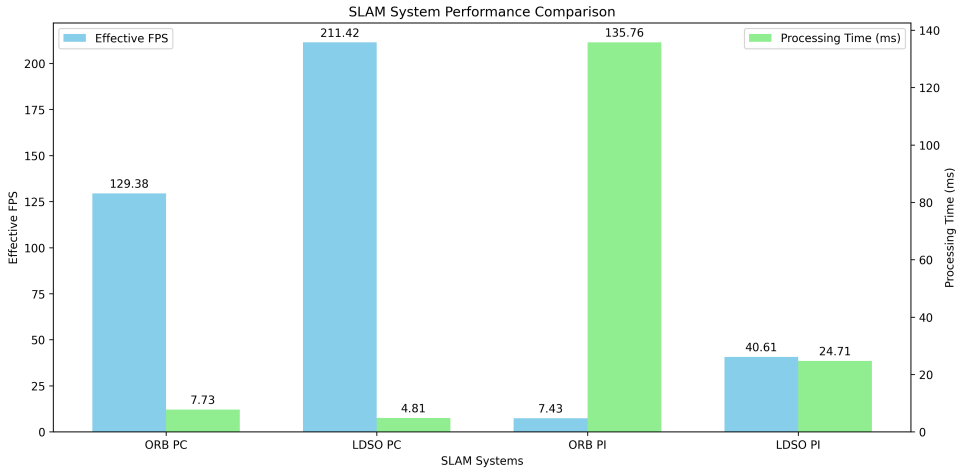


Figure 5.12: Comparison of frame rate and processing time between ORB-SLAM3 and LDSO on different platforms.

5.2.3 Closed-Loop motor Controller Testing

To test the closed-loop motor controller for forward movement the robot was instructed to move forward with no rotational component. A long exposure picture was taken over the period that the robot performed the manoeuvre and then a plane was overlayed on top of the image as well as the traced trajectory in Figure 5.13. From the plot, it is visible that the robot rejects disturbances and maintains forward movement with high levels of accuracy. The differential drive robot has a steel ball caster wheel for stability and bumps and irregularities in the driving surface cause the robot to veer off course, however each time the controller corrected the heading error to ensure it rejected disturbances as well as maintained a constant speed.

Figure 5.13: Long exposure image of straight forward movement with closed-loop control with an overlayed axis and trajectory.

Chapter 6

Conclusions and Recommendations

6.1 Conclusions

This project aimed to convert an existing robotic platform to ROS 2 and make it capable of implementing SLAM. Through different testing procedures and analysis the performance of ORB-SLAM3 and LDSO on both Raspberry Pi and PC platforms was evaluated and compared. ORB-SLAM3 outperformed LDSO in terms of accuracy and robustness and this was demonstrated on various datasets and in the real world which makes it a strong candidate as the solution to the project's requirements. ORB-SLAM3's superior performance demonstrated in this report is backed up by current research in the SLAM field [31].

Hardware choice has a significant influence on SLAM performance. The PC platform provides superior results, especially in the case of ORB-SLAM3. Real-time performance metrics demonstrated the importance of computational resources with ORB-SLAM3 on the base station (PC) processing frames at 129.38 FPS compared to just 7.43 FPS on the Raspberry Pi. ORB-SLAM3 had higher accuracy and consistency in trajectory estimation.

A common challenge for monocular SLAM systems is scale estimation and this is where ORB-SLAM3 demonstrated more consistent performance and a smaller standard deviation in the scale errors. The project successfully integrated a minimal sensor suite, including a just a USB webcam for the SLAM implementation which demonstrates the feasibility of low-cost SLAM solutions.

Based on the analysis it was concluded that ORB-SLAM3 running on a PC platform is the optimal solution for the ROS 2-based SLAM system. This conclusion is supported by its superior accuracy, exceptional real-time performance, robustness in challenging environments, consistency in scale estimation, exceptional loop closure and easy integration into ROS 2.

The implementation of a closed-loop motor controller which was not an initial project

requirement proved to be a suitable addition to the system. The testing demonstrated its effectiveness in maintaining accurate straight-line motion despite environmental disturbances. The controller's ability to reject disturbances and maintain constant speed contributes to the simplification of running SLAM in a live environment as this reduces human input when performing SLAM. This shows the importance of low-level control systems in robotic platforms and demonstrates that addressing control system problems assists the high-level system.

This project has successfully met its objectives, creating a functional ROS 2-based SLAM platform and providing valuable insights into the performance trade-offs between different SLAM algorithms and hardware configurations. The superior performance of ORB-SLAM3 on a PC platform sets a benchmark for future developments in the field of visual SLAM systems.

6.2 Recommendations

Despite the project meeting its requirements and objectives many improvements can be made on the current design in future iterations.

LDSO shows a strong potential to be run on the robot's onboard computer especially in cases where there is limited onboard compute power as it proved it has lower overhead for computation and can run at a faster rate. LDSO however has poor accuracy so to increase the accuracy an IMU could be used to aid in movement scaling and that could make it more accurate. Another option would be to use it in stereo mode however both of these would increase the amount of sensors being used and this counteracts the aim for the project which was to implement SLAM using as few sensors as possible.

ORB-SLAM3 could be further optimised for embedded systems as currently it is intended to be used for research and systems with high compute thresholds. To achieve this code optimization and hardware acceleration could greatly increase the efficiency of the algorithm on small computers. Developing more efficient feature detection algorithms to be used with ORB-SLAM3's accurate existing architecture could be of strong interest.

SLAM approaches that combine the strengths of both ORB-SLAM3 and LDSO into a single algorithm could lead to a more robust system. LDSO has shown great aptitude for systems with less compute power so this approach might yield a solution well-suited for embedded systems without compromising precision.

Currently ORB-SLAM3 as well as many other SLAM algorithms do not have sufficient support for long-term mapping. Future research could develop techniques for maintaining and updating large dynamic maps over extended periods of time.

Integrating higher-level planning, navigation and decision-making modules could enhance the robot's overall autonomy as currently there is no autonomy implementation. This

would be a practical application of the SLAM system for real-world scenarios and bridges the gap between localization and autonomous navigation.

While minimizing sensor count was a project requirement, investigating alternative sensor configurations could offer valuable insights. Evaluating the trade-offs between sensor complexity and SLAM performance, such as the benefits of stereo cameras, depth sensors, or low-cost IMUs, could inform future system designs.

A promising avenue for future research is the development of a multi-robot collaborative SLAM system. This approach would involve a swarm of robots working in together to map an environment, potentially improving mapping efficiency, coverage, and robustness. Key research challenges include developing distributed mapping algorithms, optimizing inter-robot communication protocols, and implementing effective exploration strategies. Such a system could leverage heterogeneous robot configurations, combining the strengths of different SLAM algorithms and hardware setups. This collaborative approach could significantly enhance performance in large-scale or dynamic environments, where single-robot systems face limitations in maintaining comprehensive, up-to-date maps.

These recommendations provide a roadmap for future research and development, building upon the foundations laid by this project and addressing the evolving challenges in the field of visual SLAM and autonomous robotics.

Bibliography

[1] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," *IEEE robotics automation magazine*, **vol. 13(2)**, pp. 99-110, 2006.

[2] C. Cadena et al., "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on robotics*, **vol. 32(6)**, pp. 1309-1332, 2016.

[3] M. Quigley et al., "ROS: an open-source Robot Operating System," *ICRA workshop on open source software*, **vol. 3(3.2)**, p. 5, 2009.

[4] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, **vol. 7(66)**, p. eabm6074, 2022.

[5] G. Bresson, Z. Alsayed, L. Yu, and S. Glaser, "Simultaneous localization and mapping: A survey of current trends in autonomous driving," *IEEE Transactions on Intelligent Vehicles*, **vol. 2(3)**, pp. 194-220, 2017.

[6] J. Aulinas, Y. Petillot, J. Salvi, and X. Lladó, "The SLAM problem: a survey," *Artificial Intelligence Research and Development*, pp. 363-371, 2008.

[7] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*, MIT press, 2011.

[8] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based SLAM," *IEEE Intelligent Transportation Systems Magazine*, **vol. 2(4)**, pp. 31-43, 2010.

[9] J. M. O'Kane, *A gentle introduction to ROS*, 2014.

[10] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE transactions on robotics*, **vol. 21(3)**, pp. 354-363, 2005.

[11] M. S. Tsoeu and M. Braae, "Control Systems," *IEEE*, **vol. 34(3)**, pp. 123-129, 2011.

[12] J. C. Tapson, *Instrumentation*, UCT Press, Cape Town, 2010.

[13] R. Smith, M. Self, and P. Cheeseman, "Estimating uncertain spatial relationships in robotics," *Autonomous robot vehicles*, pp. 167-193, 1990.

[14] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, **vol. 45(3)**, pp. 52-57, 2002.

[15] M. Montemerlo, "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem," *Proc. of AAAI02*, 2002.

[16] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous robots*, **vol. 4**, pp. 333-349, 1997.

[17] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g 2 o: A general framework for graph optimization," *2011 IEEE international conference on robotics and automation*, pp. 3607-3613, 2011.

[18] F. Dellaert, M. Kaess, and others, "Factor graphs for robot perception," *Foundations and Trends® in Robotics*, **vol. 6(1-2)**, pp. 1-139, 2017.

[19] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "MonoSLAM: Real-time single camera SLAM," *IEEE transactions on pattern analysis and machine intelligence*, **vol. 29(6)**, pp. 1052-1067, 2007.

[20] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," *2007 6th IEEE and ACM international symposium on mixed and augmented reality*, pp. 225-234, 2007.

[21] M. Labbé and F. Michaud, "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of field robotics*, **vol. 36(2)**, pp. 416-446, 2019.

[22] T. Pire, T. Fischer, J. Civera, P. De Cristóforis, and J. J. Berlles, "Stereo parallel tracking and mapping for robot localization," *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 1373-1378, 2015.

[23] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," *2011 10th IEEE international symposium on mixed and augmented reality*, pp. 127-136, 2011.

[24] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2D LIDAR SLAM," *2016 IEEE international conference on robotics and automation (ICRA)*, pp. 1271-1278, 2016.

[25] J. Zhang, S. Singh, and others, "LOAM: Lidar odometry and mapping in real-time," *Robotics: Science and systems*, **vol. 2(9)**, pp. 1-9, 2014.

[26] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," *2011 International conference on computer vision*, pp. 2564-2571, 2011.

[27] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, **vol. 60**, pp. 91-110, 2004.

[28] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha, "Visual simultaneous localization and mapping: a survey," *Artificial intelligence review*, **vol. 43**, pp. 55-81, 2015.

[29] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (SURF)," *Computer vision and image understanding*, **vol. 110(3)**, pp. 346-359, 2008.

[30] L. Juan and O. Gwun, "A comparison of sift, pca-sift and surf," *International Journal of Image Processing (IJIP)*, **vol. 3(4)**, pp. 143-152, 2009.

[31] C. Campos, R. Elvira, J. J. Gómez Rodríguez, J. M. M. Montiel, and J. D. Tardós, "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam," *IEEE Transactions on Robotics*, **vol. 37(6)**, pp. 1874-1890, 2021.

[32] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE transactions on Robotics*, **vol. 23(1)**, pp. 34-46, 2007.

[33] S. Macenski, F. Martín, R. White, and J. Ginés Clavero, "The marathon 2: A navigation system," *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2718-2725, 2020.

[34] T. Foote, "tf: The transform library," *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*, pp. 1-6, 2013.

[35] M. S. Bahraini, A. B. Rad, and M. Bozorg, "SLAM in dynamic environments: A deep learning approach for moving object tracking using ML-RANSAC algorithm," *Sensors*, **vol. 19(17)**, p. 3699, 2019.

[36] S. L. Bowman, N. Atanasov, K. Daniilidis, and G. J. Pappas, "Probabilistic data association for semantic slam," *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 1722-1729, 2017.

[37] B. Bescos, J. M. Fácil, J. Civera, and J. Neira, "DynaSLAM: Tracking, mapping, and inpainting in dynamic scenes," *IEEE Robotics and Automation Letters*, **vol. 3(4)**, pp. 4076-4083, 2018.

[38] M. Dymczyk, S. Lynen, T. Cieslewski, M. Bosse, R. Siegwart, and P. Furgale, "The gist of maps-summarizing experience for lifelong localization," *2015 IEEE international conference on robotics and automation (ICRA)*, pp. 2767-2773, 2015.

[39] M. Bloesch, J. Czarnowski, R. Clark, S. Leutenegger, and A. J. Davison, "Codeslam—learning a compact, optimisable representation for dense visual slam," *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2560-2568, 2018.

[40] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE transactions on pattern analysis and machine intelligence*, **vol. 40(3)**, pp. 611-625, 2017.

[41] H. Kim, S. Leutenegger, and A. J. Davison, "Real-time 3D reconstruction and 6-DoF tracking with an event camera," *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VI 14*, pp. 349-364, 2016.

[42] T. A. Vidal-Calleja, C. Berger, J. Solà, and S. Lacroix, "Large scale multiple robot visual mapping with heterogeneous landmarks in semi-structured terrain," *Robotics and Autonomous Systems*, **vol. 59(9)**, pp. 654-674, 2011.

[43] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, and others, "The limits and potentials of deep learning for robotics," *The International journal of robotics research*, **vol. 37(4-5)**, pp. 405-420, 2018.

[44] L. Carlone, G. C. Calafiore, C. Tommolillo, and F. Dellaert, "Planar pose graph optimization: Duality, optimal solutions, and verification," *IEEE Transactions on Robotics*, **vol. 32(3)**, pp. 545-565, 2016.

[45] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker, and A. J. Davison, "ElasticFusion: Dense SLAM without a pose graph," in *Robotics: Science and Systems*, **vol. 11**, p. 3, 2015.

[46] E. Karami, S. Prasad, and M. Shehata, "Image matching using SIFT, SURF, BRIEF and ORB: performance comparison for distorted images," in *arXiv preprint arXiv:1710.02726*, 2017.

[47] X. Gao, R. Wang, N. Demmel, and D. Cremers, "LDSO: Direct sparse odometry with loop closure," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 2198–2204.

[48] J. Shi et al., "Good features to track," in *1994 Proceedings of IEEE conference on computer vision and pattern recognition*, 1994, pp. 593–600.

[49] D. Gálvez-López and J. D. Tardos, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on robotics*, vol. 28, no. 5, pp. 1188–1197, 2012.

[50] M. Kaess, A. Ranganathan, and F. Dellaert, "iSAM: Incremental smoothing and mapping," *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1365–1378, 2008.

[51] A. Dwijotomo, M. A. Abdul Rahman, M. H. Mohammed Ariff, H. Zamzuri, and W. M. H. Wan Azree, "Cartographer slam method for optimization with an adaptive multi-distance scan scheduler," *Applied Sciences*, vol. 10, no. 1, p. 347, 2020.

[52] F. Dellaert, "Factor graphs and GTSAM: A hands-on introduction," *Georgia Institute of Technology, Tech. Rep*, vol. 2, p. 4, 2012.

[53] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. Kelly, and A. J. Davison, "Slam++: Simultaneous localisation and mapping at the level of objects," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2013, pp. 1352–1359.

[54] P. Niyishaka and C. Bhagvati, "Digital image forensics technique for copy-move forgery detection using dog and orb," in *Computer Vision and Graphics: International Conference, ICCVG 2018, Warsaw, Poland, September 17-19, 2018, Proceedings*. Springer, 2018, pp. 472–483.

[55] G. Lowe, "Sift-the scale invariant feature transform," *Int. J*, vol. 2, no. 91-110, p. 2, 2004.

[56] X. Liu, Y. He, J. Li, R. Yan, X. Li, and H. Huang, "A Comparative Review on Enhancing Visual Simultaneous Localization and Mapping with Deep Semantic Segmentation," *Sensors*, vol. 24, no. 11, p. 3388, 2024.

[57] L. Safarova, B. Abbyasov, T. Tsoy, H. Li, and E. Magid, "Comparison of Monocular ROS-Based Visual SLAM Methods," in *International Conference on Interactive Collaborative Robotics*, 2022, pp. 81–92.

[58] E. Lejeune, A. Queudet, S. Sarkar, and V. Lebastard, "Real-time Jitter Measurements under ROS2: the Inverted Pendulum case," unpublished.

A.1 GA Table

GA	Requirement	Justification and section in the report
1	Problem-solving	Successfully identified and formulated the engineering problem of implementing an off-the-shelf SLAM on a ROS 2 platform, as evidenced in Sections 1.1 - 1.3. Comprehensive analysis through the literature review (Chapter 2) to evaluate SLAM methodologies and algorithms. Used mathematical principles in developing control systems (Section 3.4.3) and analyzing SLAM performance (Section 5.2). The problem solution considered sustainable development through minimal sensor usage and optimization for resource-constrained systems, as outlined in Section 1.3.
4	Investigations, experiments and data analysis	Conducted thorough experimental analysis of SLAM systems using both simulation and real-world testing (Chapter 5). Performed quantitative analysis of algorithm performance through metrics like Absolute Trajectory Error (Section 5.1), processing power usage (Section 5.2.1), and scale error analysis (Section 5.2.2). Applied statistical methods to evaluate performance variations between platforms and algorithms, with results clearly presented in figures 5.1 - 5.13. Research methodology and experimental design clearly outlined in Sections 4.1 - 4.2.
5	Use of engineering tools	Demonstrated proficiency in using multiple engineering tools including ROS 2 framework (Section 4.2), OpenCV for camera calibration (Section 4.4.3), Git for version control (Appendix B.1), and various SLAM algorithms (ORB-SLAM3 and LDSO). Developed custom nodes and tools for system integration (Section 4.3). Successfully implemented and compared different SLAM algorithms using appropriate visualization and analysis tools (Section 5.2).
6	Professional and technical communication (Long report)	Produced a well-structured technical report following professional engineering standards. Included clear technical diagrams (e.g., Figures 3.1 and 4.1), mathematical equations (Sections 3.3.1, 3.4.2), and detailed system architectures. Effectively communicated complex concepts through clear writing and visual aids throughout all chapters. Created comprehensive documentation including user manuals (Appendix B) and technical specifications.

GA	Requirement	Justification and section in the report
8	Individual work	Demonstrated ability to work independently through the complete development cycle from concept to implementation. Made critical technical decisions in algorithm selection (Section 2.4), system design (Chapter 3), and implementation (Chapter 4). Managed project timeline effectively while maintaining regular supervision communication (weekly meeting and Microsoft Teams communication). Acknowledged contributions appropriately in thesis front matter while clearly delineating individual work.
9	Independent learning ability	Exhibited strong independent learning through self-directed research into SLAM methodologies (Chapter 2) and ROS 2 framework. Independently solved technical challenges in implementation (Section 4.4), developed solutions for hardware integration issues (Section 4.1), and created novel approaches for system testing (Section 5.2). Demonstrated critical thinking in analyzing results and proposing future improvements (Chapter 6).

A.2 AI Declaration

The use of large language models were used to assist with basic report writing, editing and formatting. In the initial stages of the report I required assistance on a report layout to use. I have previous reports at my disposal but prompting an LLM to produce a rough outline of the report was useful in firming my formatting technique that I was going to use throughout the report. An example of a type of prompt I asked and what output I got would be as follows:

Q: I am writing a final report for a project that has to do with the integration of an off-the-shelf SLAM solution with an existing robotic platform. The system needs to be running ROS 2 for all core functionality and needs to be able to be run on a small compute unit. Give me an outline of the format I can use for this report. Keep it scientific and compatible with IEEE conventions.

A: This is a sample output I would receive.

Title Page

Project title, author(s), affiliation, date

Abstract

150-word summary of objectives, methods, and results

I. Introduction

Problem statement Objectives System requirements

II. System Architecture

Hardware configuration Software stack overview Integration approach

III. Results

Performance metrics Resource utilization System limitations

IV. Conclusions

Summary of achievements Future work

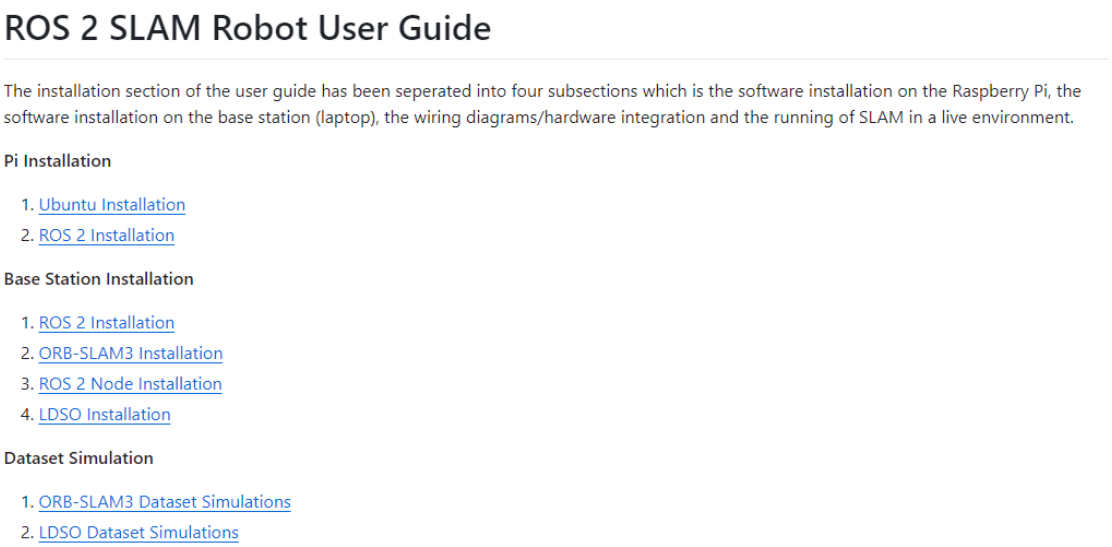
References

I would then take this advise and look at existing research papers both in UCT and in Google Scholar. Here I decided that I need an implementation section (among many things) as the LLM has omitted that in its response. This is a great example of how I used LLMs to assist me when I did not know how to proceed. This saved me crucial time when traditionally writers block would have lost me precious time in this projects already tight 12 week deadline.

B.1 GitHub

https://github.com/JosephStew-art/ROS2-SLAM-PROJECT/

B.2 User Manual



ROS 2 SLAM Robot User Guide

The installation section of the user guide has been seperated into four subsections which is the software installation on the Raspberry Pi, the software installation on the base station (laptop), the wiring diagrams/hardware integration and the running of SLAM in a live environment.

Pi Installation

1. Ubuntu Installation
2. ROS 2 Installation

Base Station Installation

1. ROS 2 Installation
2. ORB-SLAM3 Installation
3. ROS 2 Node Installation
4. LDSO Installation

Dataset Simulation

1. ORB-SLAM3 Dataset Simulations
2. LDSO Dataset Simulations

Figure B.1: ROS 2 Robot with SLAM full user guide.

For the full User Guide access it via:

User Guide GitHub

Docs/User Guide.md

B.3 ROS 2 Workspace Setup

For the full workspace setup access it via:

Workspace Setup GitHub

Docs/workspace initialisations.md

B.4 Wiring Diagram

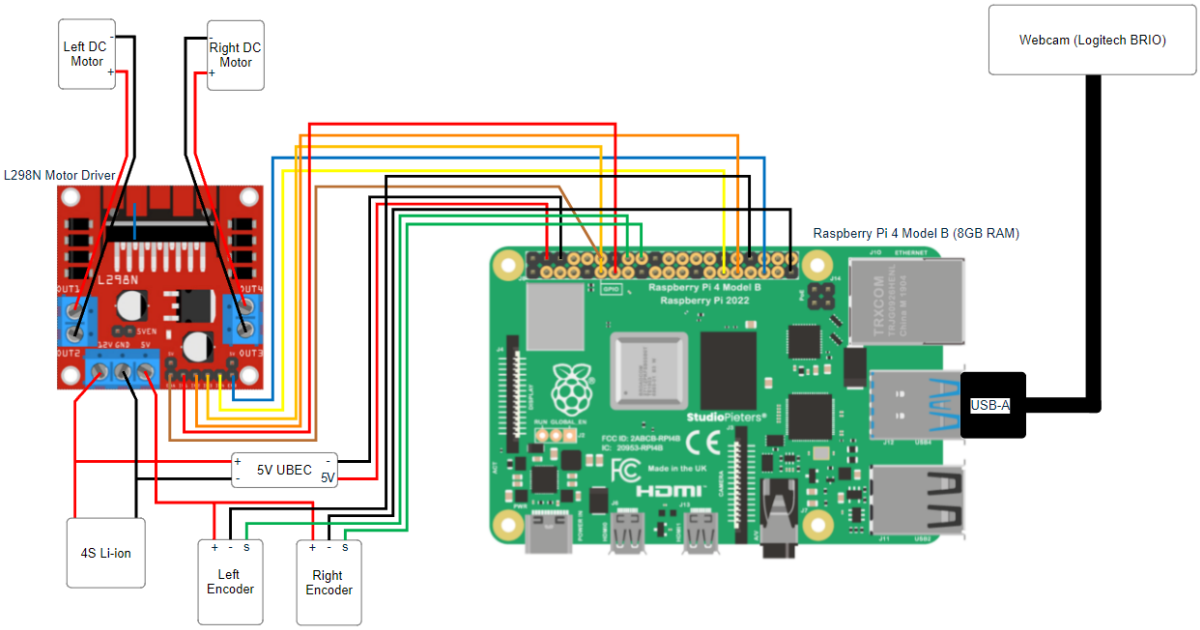


Figure B.2: Wiring diagram for the ROS 2 SLAM robot.

C.1 Camera Node

For the full camera node access it via:

Camera Node GitHub

ros2_ws/src/ros2_robot_package/ros2_robot_package/camera_publisher.py

C.2 Encoder Node

For the full encoder node access it via:

Encoder Node GitHub

ros2_ws/src/ros2_robot_package/ros2_robot_package/encoder_publisher.py

C.3 Velocity Computation Node

For the full velocity computation node access it via:

Velocity Computation Node GitHub

ros2_ws/src/ros2_robot_package/ros2_robot_package/velocity_computation.py

C.4 Motor Control Node

For the full motor control node access it via:

Motor Control Node GitHub

ros2_ws/src/ros2_robot_package/ros2_robot_package/motor_control.py

C.5 Camera Calibration Script

For the full camera calibration script access it via:

Camera Calibration Script GitHub

/cal.py

C.6 SLAM Node

The remote control and SLAM nodes are in the base station GitHub repository:

Base Station GitHub

C.6.1 Remote Control Node

For the full remote control node access it via:

Remote Control Node GitHub

orbslam3/robot_remote_control.py

C.6.2 mono.cpp

For the full mono.cpp file access it via:

mono.cpp GitHub

src/monocular/mono.cpp

C.6.3 monocular-slam-node.cpp

For the full monocular-slam-node.cpp file access it via:

monocular-slam-node.cpp GitHub

src/monocular/monocular-slam-node.cpp

C.6.4 monocular-slam-node.hpp

For the full monocular-slam-node.hpp file access it via:

monocular-slam-node.hpp GitHub

src/monocular/monocular-slam-node.hpp