YAŞAR UNIVERSITY
GRADUATE SCHOOL

MASTER THESIS

DEVELOPMENT OF AUTONOMOUS MOBILE PLATFORM USING VISUAL AND LIDAR BASED INFORMATION ON ROS ENVIRONMENT

SELİN PINAR KOÇ

THESIS ADVISOR: ASSIST. PROF. (PHD) MAHİR KUTAY

ELECTRICAL-ELECTRONICS ENGINEERING

PRESENTATION DATE: 03.08.2023

BORNOVA / İZMİR
AUGUST 2023

We certify that, as the jury, we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Jury Members:	**Signature:**
Assist. Prof. (PhD) Mahir KUTAY Yaşar University	.....
Prof. (PhD) Mustafa GÜNDÜZALP Yaşar University	.....
Assoc. Prof. (PhD) Ahmet ÖZKURT Dokuz Eylül University	.....

Prof. (PhD) Yucel Ozturkoglu
Director of the Graduate School

ABSTRACT

DEVELOPMENT OF AUTONOMOUS MOBILE PLATFORM USING VISUAL AND LIDAR BASED INFORMATION ON ROS ENVIRONMENT

Koç, Selin Pınar
MA, Electric-Electronics Engineering
Advisor: Assist. Prof. (PhD) Mahir KUTAY
August 2023

Autonomous vehicles require the integration of mechanical, computer, and electrical disciplines to achieve their intended operations. Due to their complex nature, several open-source autonomous robot platforms, including TurtleBot, ROSbot, and AZIMUT, have been specifically designed for educational and research purposes. This thesis aims to develop an affordable alternative to expensive autonomous robots for research purposes that ensure acceptable performance characteristics. The Simultaneous Localization and Mapping process is defined as gathering sensor data and creating a map while simultaneously estimating the pose of the platform. In addition, the autonomous system requires a motion-planning approach for navigation. In this thesis study, a low-cost mobile robot platform was designed, and navigation through a pre-built map while avoiding obstacles was performed. The navigation system utilizes the RTAB-Map package to incorporate a map into the navigation stack with Hector SLAM for odometry on Robot Operating System (ROS) environment. Furthermore, the Timed Elastic Band (TEB) local planner and the ROS global planner were applied, ensuring the robot's generation of efficient and safe paths. After introducing the robot platform, its limitations were discussed based on a series of experiments. The robot successfully navigated through a pre-existing map, performing obstacle avoidance along the way.

keywords: ROS, SLAM, rtabmap, hector, navigation stack, kinect, lidar

ÖZ

ROS ORTAMINDA GÖRSEL VE LIDAR TABANLI BİLGİLER KULLANARAK OTONOM MOBİL PLATFORM GELİŞTİRİLMESİ

Koç, Selin Pınar
Yüksek Lisans Tezi, Elektrik-Elektronik Mühendisliği
Danışman: Dr. Öğr. Üyesi Mahir KUTAY
Ağustos 2023

Otonom araçlar, amaçlanan işlemleri gerçekleştirmek için mekanik, bilgisayar ve elektrik disiplinlerinin bir araya getirilmesini kapsar. Karmaşık yapıları nedeniyle TurtleBot, ROSbot ve AZIMUT dahil olmak üzere çeşitli açık kaynaklı otonom robot platformları, eğitim ve araştırma amaçları için özel olarak tasarlanmıştır. Bu tez; araştırma amaçları için pahalı otonom robotlara göre uygun fiyatlı, kabul edilebilir performans özellikleri sağlayan bir alternatif geliştirmeyi amaçlamaktadır. Eşzamanlı Konumlama ve Haritalama (SLAM) süreci, eşzamanlı olarak platformun pozunu tahmin ederken sensör verilerinin toplanması ve bir harita oluşturması olarak tanımlanır. Ek olarak; otonom sistem, navigasyon için bir hareket planlama yaklaşımını gerektirir. Bu tez çalışmasında, düşük maliyetli bir mobil robot platformu tasarlanmış ve önceden oluşturulmuş bir harita üzerinden engellerden kaçınarak navigasyon gerçekleştirilmiştir. Navigasyon sistemi, Robot İşletim Sistemi (ROS) ortamında odometri için Hector SLAM ile Navigasyon Yığını'na (Navigation Stack) bir harita eklemek için RTAB-Map paketini kullanır. Ayrıca, Zamanlı Elastik Bant (TEB) yerel planlayıcısı (local planner) ve ROS küresel planlayıcısı (global planner) uygulanarak robotun verimli ve güvenli yollar oluşturması sağlandı. Robot platformu tanıtıldıktan sonra, bir dizi deneye dayalı olarak sınırlamaları tartışıldı. Robot, önceden var olan bir haritada başarılı bir şekilde gezinerek yol boyunca engellerden kaçınmayı gerçekleştirdi.

Anahtar Kelimeler: ROS, SLAM, rtabmap, hector, navigasyon yığını, kinect, lidar

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude and thanks to my advisor, Assist. Prof. (PhD) Mahir KUTAY for continuous guidance and support during this thesis process. I am honored and thankful to Yaşar University for their scholarship that has given me the opportunity to pursue my master's degree.

Selin Pınar Koç

İzmir, 2023

TEXT OF OATH

I declare and honestly confirm that my study, titled “DEVELOPMENT OF AUTONOMOUS MOBILE PLATFORM USING VISUAL AND LIDAR BASED INFORMATION ON ROS ENVIRONMENT” and presented as a Master’s Thesis, has been written without applying to any assistance inconsistent with scientific ethics and traditions. I declare, to the best of my knowledge and belief, that all content and ideas drawn directly or indirectly from external sources are indicated in the text and listed in the list of references.

Selin Pınar Koç

17.08.2023

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SYMBOLS AND ABBREVIATIONS

ABBREVIATIONS:

DoF	Degrees of Freedom
DWA	Dynamic Window Approach
EBand	Elastic Band
EKF	Extended Kalman Filter
EN	Enable
GND	Ground
GPS	Global Position System
ICP	Iterative Closest Point
IMU	Inertial Measurement Unit
IN	Input
IR	Infrared
KF	Kalman Filter
LiDAR	Light Detection and Ranging
LiPo	Lithium Polymer Battery
LTM	Long-Term Memory
UKF	Unscented Kalman Filter
PWM	Pulse Width Modulation
RANSAC	Random Sample Consensus
RBPF	Rao–Blackwellized Particle Filter
ROS	Robot Operating System
RGB-D	Red Green Blue – Depth
STM	Short-Term Memory
TF	Transform Frame

TEB	Timed Elastic Band
SLAM	Simultaneous Localization and Mapping
WM	Working Memory

SYMBOLS:

k	Time instant
u_k	Location of the vehicle
x_k	Pose of the vehicle
m_i	Location of the ith landmark
z_{ik}	Observations of the landmarks
$U_{0:k}$	Posterior control inputs for all time intervals
$Z_{0:k}$	Landmark observations for all time intervals
$\boldsymbol{b}$	Discrete trajectory
s_k	Robot poses
ΔT	Time interval
$\boldsymbol{b}^*$	Optimal trajectory
σ_i	Weight factor
P_R	Position of the mobile platform
θ	Heading angle
d	Euclidean distance

CHAPTER 1
INTRODUCTION

1.1. Overview

Lately, the field of autonomous vehicles has gained significant attention, thanks to advancements in sensor technologies. Autonomous vehicles are progressively expanding their capabilities to handle a wide array of challenging or monotonous tasks, effectively taking the place of humans in various applications, including interoffice document delivery, unmanned transportation, house cleaning, security patrolling, and more (K. C. Chen & Tsai, 2010). Navigating the mobile robots autonomously involves three main tasks: mapping (Endres et al., 2014), localization (Harik & Korsaeth, 2018), and motion planning (Marin-plaza et al., 2018). Autonomous exploration requires the creation of a precise map of its surroundings while also estimating its position on that map. Additionally, it needs to plan the most efficient path from its current location to a desired location. This involves not only finding its location but also identifying the goal position accurately on the map.

Simultaneous Localization and Mapping (SLAM) (Grisetti et al., 2010) is a step to obtain an autonomous robot. SLAM is a technique where a robot constructs a representation of its environment while simultaneously determining its position within that representation. SLAM problem was formulated by Smith and Cheeseman (1987) as a statistical estimation problem. Afterward, Smith et al. (1990) proposed the first major breakthrough in SLAM, known as Extended Kalman Filter SLAM (EKF-SLAM) that focused on a probabilistic technique to overcome the challenges posed by uncertainty in inertial sensors and measurement errors. Another essential contribution came when Durrant-Whyte et al. (1996) proved the convergence of SLAM estimates by integrating mapping and localization into a unified estimation problem.

Subsequently, SLAM solutions are categorized into online SLAM and full SLAM. Online SLAM relies on filter approaches (F. Zhang et al., 2017) including EKF-SLAM, FastSLAM and particle filters. In contrast, full SLAM uses smoothing techniques

based on least-square error minimization. One such important approach in the full SLAM category is the Graph-based SLAM, introduced by Lu and Milios (1997). This method has advantages over filtering approaches, particularly its ability to support large-scale mapping tasks while retrieving not only the recent pose and map but the complete trajectory and map. According to Das (2018b), another advantage lies in graph SLAM's enhanced precision compared to FastSLAM which is a technique that utilizes particles to approximate the robot's most probable position. Because there's a chance that the most likely location might lack a particle at any given moment, particularly in larger environments.

Despite the theoretical and conceptual developments in the SLAM algorithm, there are still notable challenges to its practical application particularly when it comes to generating and utilizing comprehensive maps. The accurate estimation of odometry to perform localization is one of these challenges. For this purpose, a laser scanner sensor was used for localization called LiDAR-based odometry as well as obstacle detection.

ROS (Quigley et al., 2009), which stands for Robot Operating System, operates as a control system for computer-managed robotic components. ROS is a go-to platform for constructing robotic systems, particularly mobile platforms with well-written documentation of the packages. Within the ROS structure, multiple individual nodes interact using a model based on subscriptions and communication orders. This versatile framework serves as a platform for crafting software for robots. It encompasses an assortment of tools, libraries, and established conventions that simplify the creation of dependable and resilient robotic behaviors across diverse robot platforms.

1.2. Problem Statement

Robot autonomous navigation, which includes motion planning and obstacle avoidance, is a highly studied area in robotic automation systems. Autonomous navigation in the ROS framework is the subject of this study. ROS offers a range of nodes and packages that facilitate complete autonomy. This thesis aims to develop a low-cost autonomous robot platform based on ROS using a combination of an RGB-D Kinect camera and an RPlidar laser scanner for perception which focuses on dynamic obstacle avoidance. The main problem is the effective utilization of LiDAR sensor data to estimate reliable odometry information for autonomous robot navigation.

It is intended to assess the ROS navigation stack with a laser scanner and a differential drive robot platform, moreover, the integration of a camera sensor into the system aims for 3D reconstruction of the environment.

1.3. Objectives

Sensor input: To obtain data from the camera and the laser scanner sensors to have an understanding of the surrounding area.

SLAM: Mapping of the unknown environment and localizing itself just with the laser scanner without wheel encoder, IMU, or GPS.

Path Planning: Estimating a safe and optimal path that avoids obstacles and dynamically adjusts as the environment changes

Control: Moving the platform according to the planned trajectory without collision.

1.4. Thesis Outline

The subsequent chapter delivers an overview of the theoretical background and literature research. Within this section, the thesis explains the ROS framework, its historical context, and its notable features. Additionally, the RTAB-Map and Hector SLAM packages, along with the ROS Navigation package utilized for the navigation of the robot, are provided. Afterward, Chapter 3 presents the overall design of the robot platform. Subsequently, Chapter 4 details the implementation of packages within the ROS environment. In Chapter 5, the experimental outcomes of the navigation process are presented. Lastly, Chapter 6 serves to summarize all the acquired results.

CHAPTER 2
BACKGROUND AND LITERATURE REVIEW

The chapter of this thesis presents the background and literature review section which focuses on key headings including ROS, SLAM packages, and ROS Navigation Stack. The chapter establishes a foundation for the subsequent sections of the thesis by presenting an overview of each topic and highlighting their significance to achieving autonomous robot navigation.

2.1. Robot Operating System (ROS)

ROS is a Linux-based open-source framework to develop robot applications. The ROS is a middleware that provides a suite of software resources, including libraries and tools. It has the capabilities of passing messages between processes, supporting common programming languages, and enabling third-party libraries like OpenCV and Point Cloud Library. Many popular robotic algorithms have still been implemented, such algorithms are SLAM and motion planning. Additionally, the ROS provides package customization to any robot since most of the existing packages are open source. Therefore, the time spent developing robot software is significantly reduced (Joseph, 2018).

The history of the ROS dates back to its inception in 2007 at Stanford University, when an ensemble of software tools was initially created for robots by Morgan Quigly. Later that year, the project transitioned to Willow Garage, a robotics research startup, that coined the Robot Operating System. In subsequent years the release of various ROS versions was witnessed, including significant milestones such as the development of a functional ROS robot called PR2 (Cousins, 2010) and the introduction of ROS 1.0, which contained enduring features still in use today. ROS versions with distinct names like “Indigo Igloo” (the first long-term support release), “Jade Turtle”, “Kinetic Kame”, “Lunar Loggerhead”, “Melodic Morenia” and "Noetic Ninjemys" followed in subsequent years, each contributing to the advancement of the Robot Operating System. (Joseph, 2018).

2.1.1. ROS Architecture and Concepts

A brief explanation of the ROS architecture and important concepts of ROS is displayed in the following. Nodes, messages, topics, and services are the core principles of ROS implementation. In general, ROS sets an inter-process communication to send data between two independent programs which are represented as nodes in Figure 2.1. Publisher nodes are the nodes that send data and subscriber nodes are the nodes that receive data. Nodes connect through a ROS master to communicate with each other by sending the information to the ROS master as well as its data type. The nodes connect through a ROS master to communicate with each other by sending the information and data type to the ROS master. In the ROS framework, nodes exchange messages through ROS topics, which serve as a communication channel. These messages can take different formats such as integers, floats, and strings. Additionally, the ROS service follows a Request/Reply mechanism similar to the Publish/Subscribe approach used in ROS topics. The node that creates the service is referred to as the server node, while the node that makes the service request acts as the client node.

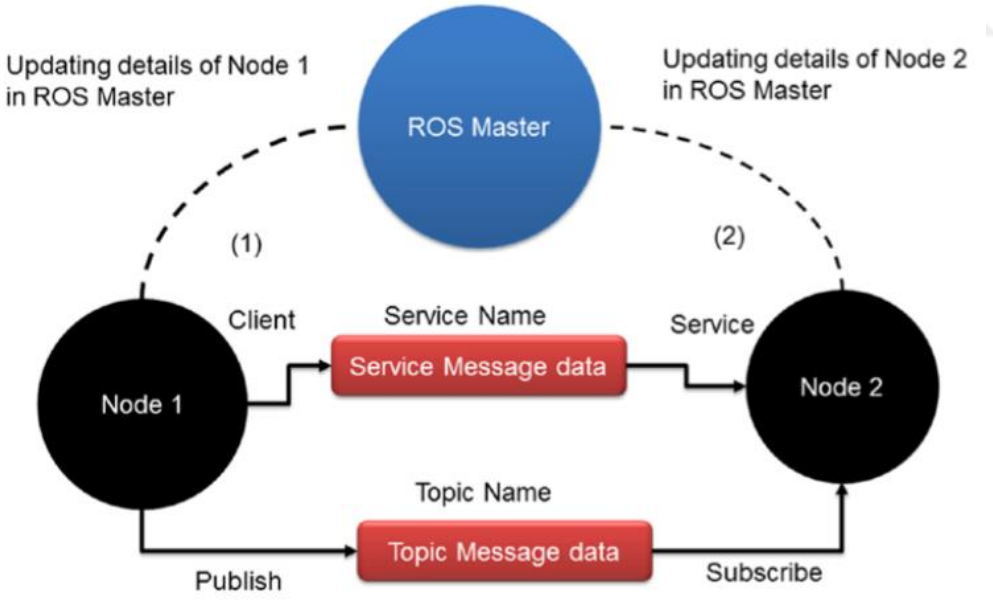


Figure 2.1. Communication Block Diagram of ROS (Joseph, 2018)

A service called “roscore” plays a crucial role in the ROS by facilitating the connection between nodes and enabling the transmission of messages. During startup, each node connects to “roscore” for registering information about the message streams it publishes and the streams it intends to subscribe to. When a new node joins, “roscore”

provides the necessary details for establishing direct peer-to-peer connections with other nodes that are publishing or subscribing to the same message topics. Running “roscore” is essential for every ROS system as it enables the discovery and communication between nodes. Without “roscore”, nodes would not be able to locate and connect with other nodes in the system (Morgan Quigley, Brian Gerkey, 2015).

Before to start developing with ROS, a workspace called catkin is needed be set which is a place to store the ROS packages. It serves as a build system responsible for generating executable programs, libraries, and scripts. A catkin workspace consists of three essential directories: “src”, “build”, and “devel” with each serving a specific purpose.

2.1.2. Visualization and Monitoring

Rviz serves as a 3D visualization tool within the ROS that can display sensor data from the camera as a point cloud and from LiDAR as a laser scan as well as the map produced by the robot. Additionally, the user can observe the robot’s current position and the trajectories drawn on the map when a target location is set. The “rqt_graph” is a visual tool for inspecting and monitoring the running processes and their relations with each other with a tree structure on ROS. The nodes of this tree consist of nodes running on ROS which connects. The "roslaunch" command has the purpose of executing multiple nodes simultaneously through the utilization of launch files within the ROS environment. Consequently, the need of initiating separate terminal screens and repeatedly input commands for each node is prevented.

2.1.3. Coordinate Frames

Coordinate frames must be set to provide the transformation of data from one frame to another. A standardized convention for coordinate frames of a mobile robot is provided in Figure 2.2. The LiDAR based on the platform measures the distance to objects in its coordinate frame. However, it is necessary to convert the detected object's coordinates from the LiDAR's frame to the robot base's frame to achieve effective navigation and obstacle avoidance.

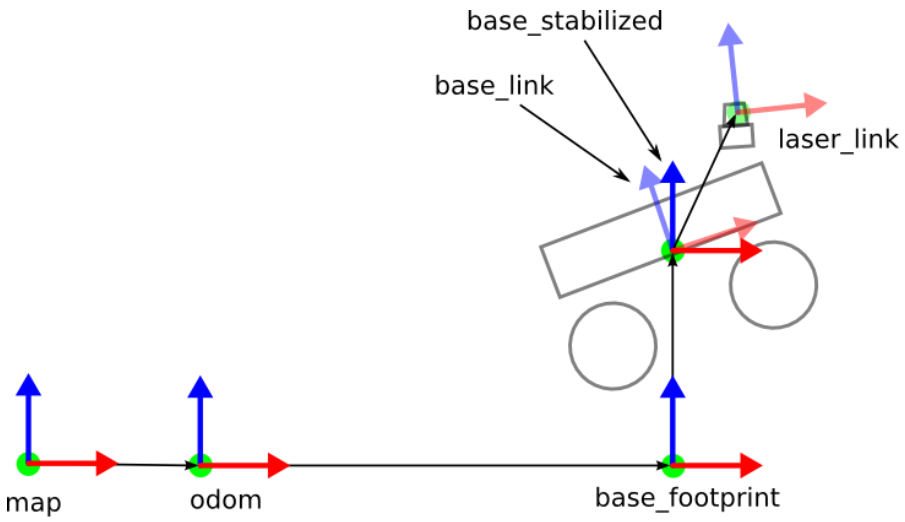


Figure 2.2. Coordinate Frames of a Mobile Robot (StefanKohlbrecher, 2011)

- The "map" frame offers a global reference for determining the robot's position within its environment. It remains fixed and denotes the absolute positioning of objects in the environment.
- The "odom" frame is a fixed frame with respect to the world, derived from its internal motion sensors, often wheel encoders. It establishes a local frame of reference that is relevant to the robot's initial position, allowing for tracking of its movement over time.
- The "base_footprint" frame corresponds to the physical base or chassis of the robot. Positioned at the central point of the robot's chassis, it acts as the origin for the local coordinate system, contributing to spatial determination.
- The "base_link" frame is attached to the robot's base, co-moving with it. It characterizes the robot's position and orientation relative to the "base_footprint" frame. Typically, it serves as a parent frame for other constituent elements of the robot.
- The "laser_link" frame designates the coordinate frame associated with the robot's laser sensor which portrays the pose of the laser sensor in relation to the robot's base represented by the "base_link" frame. Laser scan data is conventionally expressed within the "laser_link" frame.

2.2. Simultaneous Localization and Mapping (SLAM)

Simultaneous Localization and Mapping (SLAM) (Bailey & Durrant-Whyte, 2006; Durrant-Whyte & Bailey, 2006) refers to a process where a robot dynamically builds a map of its environment while simultaneously estimating its own position within that map. The SLAM problem is a necessary step to receive a fully autonomous robot and has been one of the interests of researchers in the field of robotics. Figure 2.3 illustrates a scenario where a mobile robot navigates in an unfamiliar area while simultaneously observing and recording the relative movement of unknown landmarks through sensors. Subsequently, the determination of the landmarks' relative positions and the mobile platform's pose are performed in real-time.

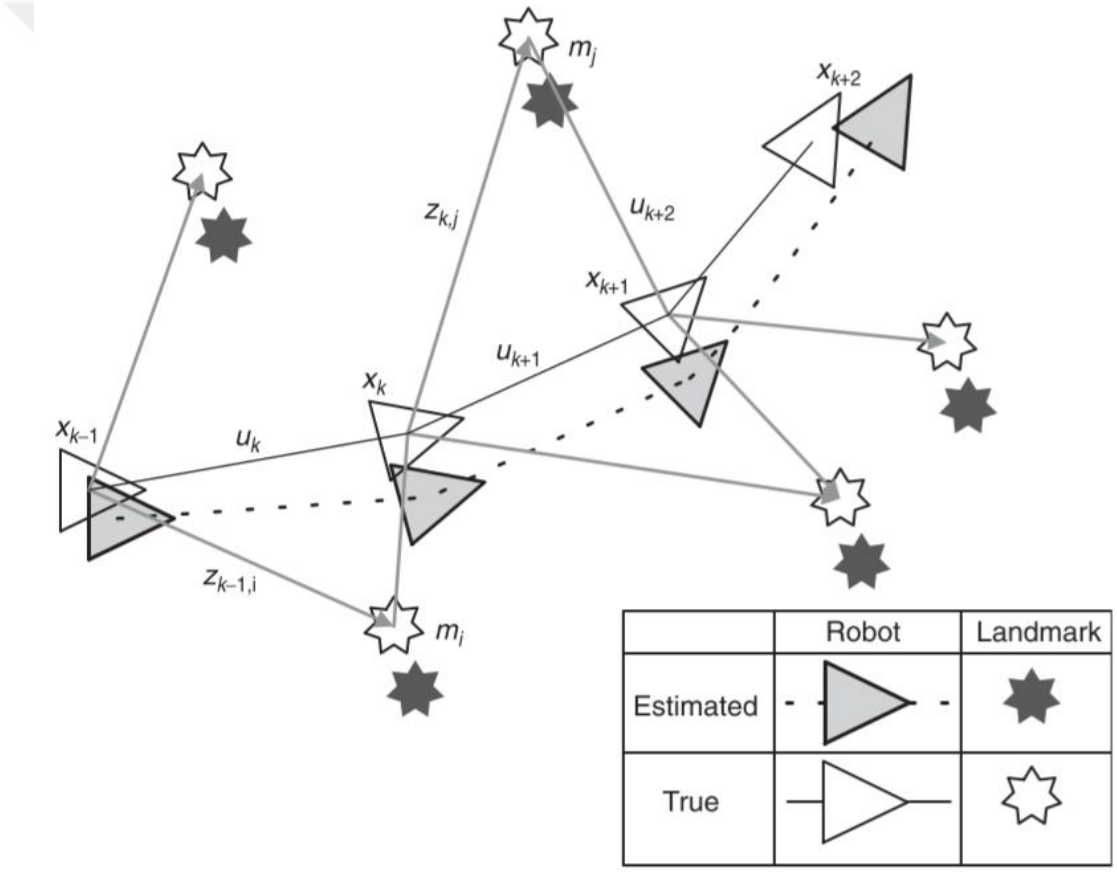


Figure 2.3. The SLAM Problem with True and Estimated Locations (Durrant-Whyte & Bailey, 2006)

In the figure below, $\boldsymbol{x}_k$ represents pose of the vehicle, $\boldsymbol{u}_k$ is the motion vector of the vehicle between poses. Location of the i[th] landmark are represented as $\boldsymbol{m}_i$ and observations taken from these landmarks are given as $\boldsymbol{z}_{ik}$.

The probabilistic distribution of the SLAM problem which are necessary to compute for all times k given:

$$P(\boldsymbol{x}_k, \boldsymbol{m} | \boldsymbol{Z}_{0:k}, \boldsymbol{U}_{0:k}, \boldsymbol{x}_0) \quad (1)$$

The probability distribution highlights the need of posterior landmark observations of all time intervals ($\boldsymbol{Z}_{0:k}$), the posterior control inputs of all time intervals ($\boldsymbol{U}_{0:k}$), and the previous pose of the vehicle which is crucial for determining the current vehicle state, denoted as $\boldsymbol{x}_k$, and locations of the landmarks, denoted as $\mathbf{m}$.

The observed data and prior knowledge are considered to evaluate the uncertainty related to the vehicle's position and the map. It estimates and updates the system state in SLAM algorithms, enabling the robot to refine its understanding of its position and the surrounding environment.

This distribution captures the level of uncertainty or likelihood associated with the previous pose of the vehicle and the environment map. It takes into account all the information available up to time step $k-1$. Also, it serves for recursively estimating and modifying the existing state estimate at time k in the SLAM process. The estimation starts at $k-1$ time for the distribution $P(\boldsymbol{x}_{k-1}, \boldsymbol{m} | \boldsymbol{Z}_{0:k-1}, \boldsymbol{U}_{0:k-1})$.

The two model are required for defining this calculation which are an observation model and a state transition model. Equation (2) is the observation model which defines the probability of the observations ($\boldsymbol{z}_k$) when the vehicle state and locations of the landmark are known.

$$P(\boldsymbol{z}_k | \boldsymbol{x}_k, \boldsymbol{m}) \quad (2)$$

The motion model in (3) defines the probability distribution of state transitions that follows a Markov process, where the subsequent state ($\boldsymbol{x}_k$) is solely dependent on the preceding state ($\boldsymbol{x}_{k-1}$) and the applied control ($\boldsymbol{u}_k$). Importantly, this transition is assumed to be independent of observations and the map, ensuring that the model captures the dynamics of the vehicle's movement regardless of the surrounding environment.

$$p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \tag{3}$$

One of the solutions of SLAM is the EKF-SLAM method which is presented below. The equation expresses the probability distribution that characterizes where the vehicle is likely positioned as

$$P(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \Leftrightarrow \boldsymbol{x}_k = \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) + \boldsymbol{w}_k \tag{4}$$

where function $\boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k)$ represents the vehicle's state changes based on its previous state $\boldsymbol{x}_{k-1}$ and the control input $\boldsymbol{u}_k$ while the term $\boldsymbol{w}_k$ signifies motion disturbances which follow a Gaussian distribution with a mean of zero with covariance $\boldsymbol{Q}_k$.

$$P(\boldsymbol{z}_k|\boldsymbol{x}_k, \boldsymbol{m}) \Leftrightarrow \boldsymbol{z}_k = \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{m}) + \boldsymbol{v}_k \tag{5}$$

where $\boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{m})$ represents the observation model, describing how the expected measurement relates to the vehicle's state $\boldsymbol{x}_k$ and the map $\boldsymbol{m}$. The term $\boldsymbol{v}_k$ accounts for Gaussian-distributed, uncorrelated, and possesses zero mean observation errors with covariance $\boldsymbol{R}_k$. The next equation represents the mean compute obtained using the EKF method as

$$\begin{bmatrix} \hat{\boldsymbol{x}}_{k|k} \\ \hat{\boldsymbol{m}}_k \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_k \\ \boldsymbol{m} \end{bmatrix} | \boldsymbol{Z}_{0:k} \tag{6}$$

The covariance update refines the uncertainty estimates as

$$\boldsymbol{P}_{k|k} = \begin{bmatrix} \boldsymbol{P}_{xx} & \boldsymbol{P}_{xm} \\ \boldsymbol{P}_{\boldsymbol{xm}}^T & \boldsymbol{P}_{mm} \end{bmatrix}_{k|k} = E\left[\begin{pmatrix} \boldsymbol{x}_k - \hat{\boldsymbol{x}}_k \\ \boldsymbol{m} - \hat{\boldsymbol{m}}_k \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_k - \hat{\boldsymbol{x}}_k \\ \boldsymbol{m} - \hat{\boldsymbol{m}}_k \end{pmatrix}^T | \boldsymbol{Z}_{0:k} \right], \tag{7}$$

The “Time Update” equations represent the estimation of the state and covariance prediction as

$$\hat{\boldsymbol{x}}_{k|k-1} = \boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_k) \tag{8}$$

$$\boldsymbol{P}_{xx,k|k-1} = \nabla \boldsymbol{f} \boldsymbol{P}_{xx,k-1|k-1} \nabla \boldsymbol{f}^{\boldsymbol{T}} + \boldsymbol{Q}_k \tag{9}$$

Where $\nabla \boldsymbol{f}$ denotes the Jacobian matrix of the state transition function $\boldsymbol{f}$.

The "Observation Update" equations which update the state and map estimates using the predicted values represented as

$$\begin{bmatrix} \hat{\boldsymbol{x}}_{k|k} \\ \hat{\boldsymbol{m}}_k \end{bmatrix} = \left[\hat{\boldsymbol{x}}_{k|k-1} \hat{\boldsymbol{m}}_{k-1} \right] + \boldsymbol{W}_k \left[\boldsymbol{z}_k - \boldsymbol{h}\left(\hat{\boldsymbol{x}}_{k|k-1}, \hat{\boldsymbol{m}}_{k-1} \right) \right] \tag{10}$$

$$\boldsymbol{P}_{k|k} = \boldsymbol{P}_{k|k-1} - \boldsymbol{W}_k \boldsymbol{S}_k \boldsymbol{W}_k^{\boldsymbol{T}}, \tag{11}$$

where

$$\boldsymbol{S}_k = \nabla \boldsymbol{h} \boldsymbol{P}_{k|k-1} \nabla \boldsymbol{h}^T \boldsymbol{R}_k \tag{12}$$

$$\boldsymbol{W}_k = \boldsymbol{P}_{k|k-1} \nabla \boldsymbol{h}^T \boldsymbol{S}_k^{-1} \tag{13}$$

which the symbol $\nabla \boldsymbol{h}$ represents the Jacobian matrix of function $\boldsymbol{h}$, assessed at the estimated state $\hat{\boldsymbol{x}}_{k|k-1}$ and the previous map estimate $\hat{\boldsymbol{m}}_{k-1}$.

Implementing a SLAM system has the frontend and the backend components as given in Figure 2.4. In Visual SLAM, the front end involves detecting and tracking features from imaging sensors, such as monocular or stereoscopic cameras, using a technique called structure from motion. In LiDAR-based SLAM, the frontend involves scanning the environment and registering successive point clouds using scan matching techniques like Iterative Closest Point (ICP). Thus, the features can be sparse landmarks or dense point clouds, depending on the sensor and detection method used.

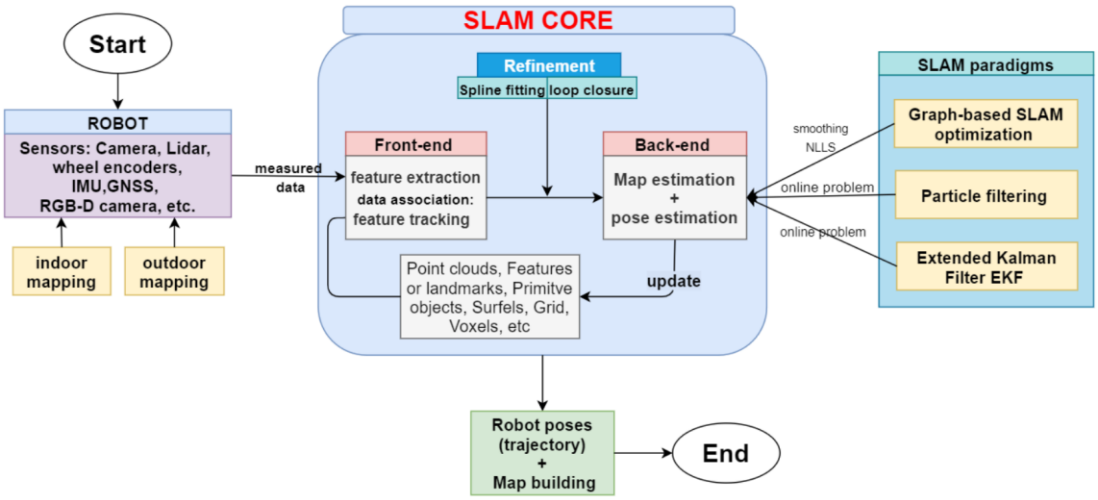


Figure 2.4. Overview of SLAM Techniques (Alsadik & Karam, 2021)

Different methods are used to calculate the robot's pose (position and orientation) based on its motion, known as odometry. These methods include wheel encoders, which track wheel rotations; Inertial Measurement Units (IMUs), which use sensors to measure linear and angular motion; visual odometry, which analyzes visual features for motion estimation; LiDAR odometry, which utilizes laser measurements; and sensor fusion, which combines multiple sensors for enhanced accuracy. Each method has its unique strengths and constraints, and the choice depends on factors like the robot's purpose and the surrounding environment.

The success of feature-based SLAM algorithms relies heavily on the frontend's ability to accurately detect and track features. However, challenges arise in textureless or poorly textured areas for image-based SLAM, as well as environments lacking planar structures for future-based SLAM. Outliers in the frontend can significantly affect the performance of the backend, which estimates the robot's pose and landmark locations. The backend using mathematical techniques like filtering or smoothing focuses on updating these estimates in a reference coordinate system.

Numerous algorithms have been created to address the SLAM problem. Bresson et al. (2017) categorized SLAM estimation techniques as filter-based approach and optimization-based approach. Filter-based methods involve a two-step iterative process derived from Bayesian filtering. In the first step, the robot's pose and map are predicted, and then, the predicted map is corrected by matching the current map. Filter-based methods include Extended Kalman Filter (EKF) (Nurmaini & Pangidoan, 2018), Unscented Kalman Filter (UKF) (Ullah et al., 2020), Information Filter (IF), and

Particle Filter (PF) (Talwar & Jung, 2019). Even though optimization-based approaches localize highly accurately and require less memory, the filter-based approach needs less computing source (C. Chen et al., 2018).

The Kalman Filter (KF) is primarily designed for linear systems and is not commonly used in SLAM applications. In contrast, the EKF is widely employed in SLAM because it is specifically designed for handling non-linear systems. The UKF, introduced by Julier & Uhlmann (2004), is a linear estimator that eliminates the need for linearization in non-linear systems, as required by the EKF. The UKF offers a simpler and more accurate alternative to the EKF; while maintaining performance comparable to the KF. The IF is the inverse version of the KF, but it is not as widely preferred in mono-vehicle SLAM as the EKF because it necessitates converting each measurement into its inverse form, which can be computationally expensive. Nevertheless, the IF has found extensive use in multi-vehicle SLAM applications (Bresson et al., 2017).

One of the most popular SLAM is Gmapping algorithm (Grisetti et al., 2007) which is a laser-based SLAM based on a PF paring algorithm whereas Hector SLAM utilizes scan matching algorithm altering as does not depend on odometer data. According to the Gmapping algorithm, the state transition function is predicted using an algorithm called Rao–Blackwellized particle filtering (RBPF) (X. Zhang et al., 2020).

Xbox Kinect was used by Oliver et al. (2012) to perform 2D and 3D SLAM and navigation applications. The 3D model of the surroundings was constructed with point cloud data by performing 3D SLAM. The projection of the point cloud data onto a 2D plane enables the application of a 2D SLAM algorithm. The proposed solution is applicable but requires higher processing as well as its field of view and range are limited.

2.2.1. Hector SLAM

Hector SLAM (Kohlbrecher et al., 2011), is an algorithm that matches the scans and estimates a robot's pose using 2D LiDAR data using high sampling frequency proposed by a team at the Technical University of Darmstadt. The algorithm deals with discrete occupancy grid maps, which have limitations in precision and interpolation capabilities. It employs a technique called Gaussian-Newton to address the problem of scan matching and maps laser points to the existing map's rigid body transformation.

Scan matching is a crucial process for aligning laser scans with an existing map or each other. The procedure starts by employing a bilinear function to estimate the probability of a grid being occupied and then proceeds to divide the grid map accordingly. The objective is to obtain the optimal rigid transformation $\boldsymbol{\xi} = (p_x, p_y, \psi)^T$ by minimizing the total error for the best scan alignment with the map as

$$\boldsymbol{\xi}^* = \underset{\xi}{\operatorname{argmin}} \sum_{i=1}^{n} [1 - M(\boldsymbol{S_i}(\boldsymbol{\xi}))]^2 \tag{14}$$

where the grid occupancy is $M(\boldsymbol{S_i}(\boldsymbol{\xi}))$ while $\boldsymbol{S_i}(\boldsymbol{\xi})$ is the endpoint of the laser that has the coordinates $\boldsymbol{S_i}(\boldsymbol{S_{ix}}, \boldsymbol{S_{iy}})$ in global coordinate systems as

$$\boldsymbol{S_i}(\boldsymbol{\xi}) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \sin\theta \end{bmatrix} \begin{bmatrix} \boldsymbol{S_{ix}} \\ \boldsymbol{S_{iy}} \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix} \tag{15}$$

As the transformation ξ is continued to optimize by making small adjustments ($\Delta\xi$), the sum of squared errors tends to approach zero.

$$\sum_{i=1}^{n} [1 - M[\boldsymbol{S_i}(\boldsymbol{\xi} + \Delta\boldsymbol{\xi})]]^2 \to 0 \tag{16}$$

A node called “hector_mapping” was preferred for LiDAR-based SLAM since it requires low computational resources. The pose of the robot platform was estimated at the scan rate of the RPlidar laser scanner (10 Hz). The “hector_mapping” produces a grid map from LiDAR measurements using the scan matching algorithm (Grisetti et al., 2007) based on the ICP (Donoso et al., 2017).

Initially, the algorithm goes through iterations with the starting position estimated by the odometer. It then matches the scan data to generate a more precise and computationally efficient map (X. Zhang et al., 2023). The algorithm incorporates state estimation in navigation by filtering inertial measurement system (IMU) data using EKF. Hector SLAM is particularly effective in small indoor environments with high-frequency 2D LiDARs Additionally, it is suitable for robots with limited computing capabilities as it does not rely heavily on odometry data.

2.2.2. RTAB-MAP SLAM

Real-Time Appearance-Based Mapping (RTAB-Map) is a graph-based SLAM technique that utilizes a loop closure process. It has an implementation for both LiDAR-based or visual-based SLAM and a variety of setups for 2D or 3D solutions. The structure of the RTAB-Map SLAM package is represented in Figure 2.5 which contains “rtabmap” as the main node.

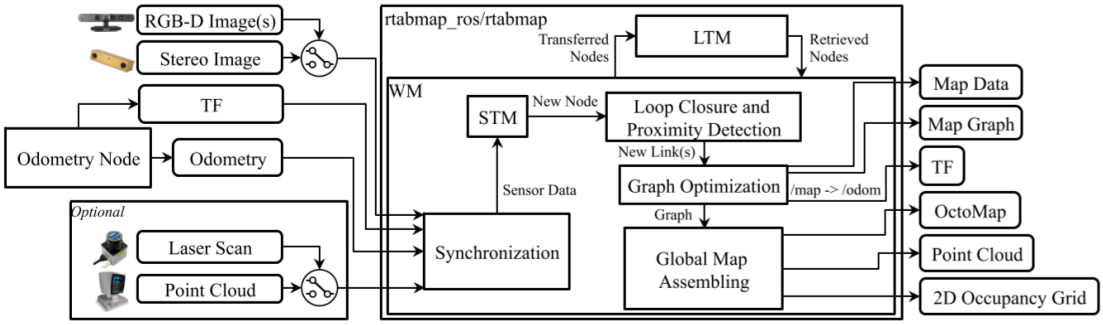


Figure 2.5. RTAB-Map Block Diagram (Labbé & Michaud, 2018b)

Required inputs are camera images from either RGB-D or stereo camera, TF which specify the sensor locations according to the robot base and odometry information. Because the odometry integrated as an external source, it can be from any kind of source suitable for the robot and its application. Instead of odometry from wheel encoders and IMU, odometry based on visual or LiDAR inputs can be implemented into the system. Laser scan or point cloud are the optional inputs. The sensor synchronization is applied to all sensor messages, then the Short-Term Memory (STM) module uses the pose of the odometry, raw data from sensor and some supplementary information to create a new node. The loop closure detection method can identify whether the recent location is novel or has been visited before, regardless of the robot’s position (Labbé & Michaud, 2018a). Loop closure detection is often accomplished using appearance-based approaches (Garcia-Fidalgo & Ortiz, 2015), which compares images for distinct features.

A link between nodes can preserve a rigid transformation. Within this system, three types of links are Neighbor link, Loop Closure link, and Proximity link. Neighbor connections are employed in the STM to link a new node with the previous node. Loop closure link is established through loop closure detection techniques. The proximity link, on the other hand, is created using LiDAR data and scan alignment methods. The links serve as constraints in graph optimization, which reduces odometry drift by

propagating the computed error throughout the entire graph when new loop closures or proximity links are introduced into the graph. After the optimization process is completed, the OctoMap, Point Cloud, and 2D Occupancy Grid outputs are combined and prepared to publish. Additionally, the TF transformation enables odometry correction, which determines the robot's location in the map frame (Labbé & Michaud, 2018b).

Memory management technique of RTAB-Map proposed by Labbé & Michaud (2018a). ensures the efficient operation of graph management modules, enabling long-term online SLAM in large environments. By controlling the graph size, it prevents processing time from exceeding real-time constraints, which can occur when modules are processing a growing graph. Working Memory (WM), Long-Term Memory (LTM), and STM construct the memory management of RTAB-Map. When the update time surpasses a predetermined threshold, the WM is transferred to the LTM, resulting in a reduction in WM size and an improvement in update time. Upon the creation of a new node, it undergoes a visual comparison with the last node. If they show similarities, an increment of the new node's weight occurs based on the previous node. In situations where certain thresholds related to time or memory are reached, the nodes with the least weight are moved to the LTM. Retrieving the neighboring nodes of the corresponding location in the WM from the LTM enables additional loop closure and proximity detections. This incremental process enables the robot to expand the map and utilize previous places for localization as it traverses previously explored areas.

2.3. ROS Navigation Stack

Navigation Stack is specifically designed for 2D navigation tasks, performs functions such as planning paths and sending instructions for movement to the mobile base of the robot. Figure 2.6 represents the "move_base" components, along with their interactions within the ROS system. The nodes that have white background are the provided nodes by the navigation stack. The platform-specific nodes are represented as light gray with dashed lines is the specific nodes that are provided to the "move_base". The diagram includes two additional nodes, namely "amcl" and "map_server," depicted as dark gray nodes with dashed lines. These nodes are optional and can be utilized as part of the system.

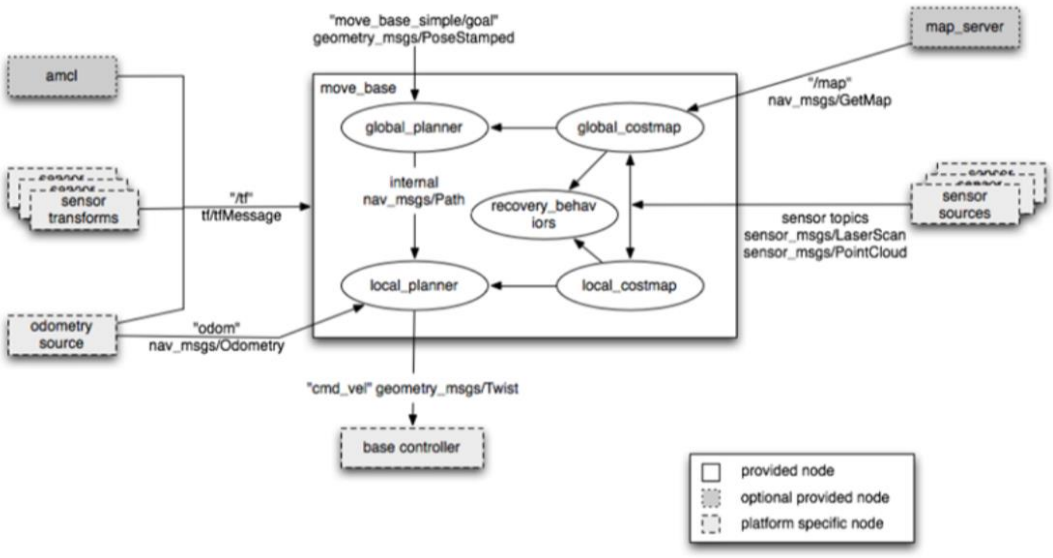


Figure 2.6. Navigation Stack Overview (Guimarães et al., 2016)

Navigation stack can be used with two implementation methods: publishing the constructed map using “map_server”, publishing map while constructing simultaneously in navigation using “gMapping” or “hector_mapping”. The “amcl” package is a localization system called Adaptive Monte Carlo Localization (AMCL) which needs a pre-build map and the starting position. In the AMCL method, particles are randomly distributed to the possible locations and the robot’ pose is predicted by a particle filter (Das, 2018a). In short, “/amcl” and “/map_server” nodes are used for integrating the static map into the “move_base”. For 2D translation and rotation in the xy-plane, the navigation stack publishes velocity commands with the topic name "cmd_vel".

A costmap consists of three layers which are named as static map, obstacle map, and inflation. The provided map forms the layer of the static map, while obstacles form the layer of the obstacle map. The layer of inflation incorporates inflated obstacles, which involves expanding the area around detected obstacles within a specified radius. This inflation assigns a certain cost ranging from 0 to 255 which are categorized as occupied, free, or unknown according to the obstacles (Zheng, 2021). Local path planning focuses on obstacle avoidance, while global path planning aims to find the shortest path. Combining both planning algorithms ensures accurate robot navigation. Handling deviations caused by dynamic surroundings requires the integration of local path planning (X. Zhang et al., 2020).

The tree global planner “carrot_planner”, “navfn”, and “global_planner” are generally used in ROS navigation. “carrot_planner” is a basic method that guides the robot at a close point to the goal if there is an obstacle at that goal. “navfn” finds a path with least possible cost using Dijkstra’s algorithm and “global_planner” has more algorithm alternatives like Dijkstra’s algorithm or A∗ algorithm (Zheng, 2021).

The chosen algorithm for the global planner is Dijkstra because it is easy to debug and performs well in experiments. The approach of Dijkstra's algorithm is employed a grid cell map that assigns cost values to candidate nodes for navigation. The shortest path to the destination is determined by incrementing the cost values based on the number of nodes required to reach each node (Marin-plaza et al., 2018).

Trajectory Rollout and DWA are the default local planners provided by the “base_local_planner” package in ROS. Comparing local planners requires consideration of different criteria for each planner. The DWA (Fox et al., 1997) performs well in terms of smooth motion, and repeatability. In contrast, the Elastic bands (EBand) (Quinlan & Oussama, 1993) demonstrates higher accuracy in reaching the goal. The TEB local planner (Rösmann et al., 2012) excels in obstacle avoidance and reactivity. Thus, for a differential drive robot, the TEB local planner is a favorable choice for obstacle avoidance (Naotunna & Wongratanaphisan, 2020).

2.3.1. TEB Local Planner

The trajectories generated using TEB are represented as discrete sequences of poses, each associated with a specific time stamp. A discrete trajectory $\boldsymbol{b}$ represents as

$$\mathbf{b}= [\boldsymbol{s}_1, \Delta T_1, \boldsymbol{s}_2, \Delta T_2, \dots, \boldsymbol{s}_{N-1}, \Delta T_{N-1}]^{\mathrm{T}} \tag{17}$$

It consists of an ordered sequence of poses and time intervals. Each pose, denoted as $s_k = [x_k, y_k, \beta_k] \in R^2 \times S^1$ with $k = 1, 2, \dots, N$. These poses consist of the robot pose (x_k, y_k) and orientation (β_k). The time intervals, denoted as $\Delta T_k \in R_{>0}$ with $k = 1, 2, \dots, N-1$, specify the duration of movement between one pose s_k to another s_{k+1}. A representation of a trajectory that includes three poses is displayed in Figure 2.7, where the reference frame is denoted as {map}.

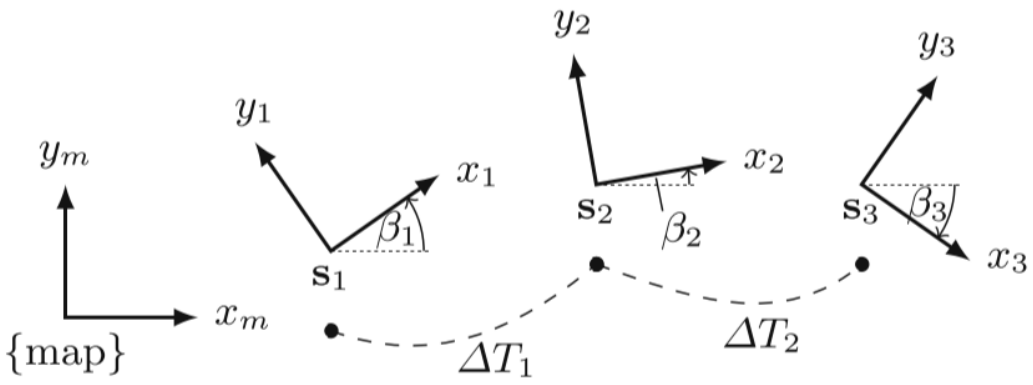


Figure 2.7. Discrete Path with n=3 Poses (Albers et al., 2019).

Time Elastic Bands (TEB) optimization problem is focused on finding the optimal trajectory, denoted as $\boldsymbol{b}*$, that meets the motion planning requirements by minimizing a cost function comprising various objectives and constraints. The optimization problem in the context of TEB involves a cost function that considers different sets of objectives (J) and penalties (P). Each objective and penalty is assigned a weight factor (σ_i) as:

$$\boldsymbol{b}^* = \underset{\boldsymbol{b}\setminus\{\boldsymbol{s}_1,\, \boldsymbol{s}_N\}}{\operatorname{argmin}} \sum_i \sigma_i f_i^2(\boldsymbol{b}), \qquad i \in \{\mathcal{J}, \mathcal{P}\} \tag{18}$$

In order to adjust to the changing environment, a control scheme called model predictive is employed to refine the trajectory. Solving the problem of optimization is performed at the interval of each sampling repeatedly, considering the current robot pose and velocity. Only the first control action from the computed trajectory is executed at each time step, following the principles of model predictive control (Albers et al., 2019).

CHAPTER 3
ROBOT PLATFORM

The robot platform developed within the scope of this thesis has 4 wheels controlled by differential drive and a motor driver with a dual H bridge. Since the selected motor driver can handle two DC motors simultaneously, the front and rear motors connected to the wheels of the robot work synchronously with each other. The microcontroller and battery required for the movement of the robot were placed on the interlayer of the platform. A laser distance measurement sensor capable of scanning at an angle of 360°, a RGB-D camera and a portable computer were positioned on the top layer of the robot. The constructed robot platform is given in Figure 3.1 and the robot platform specifications are presented in Table 3.1.

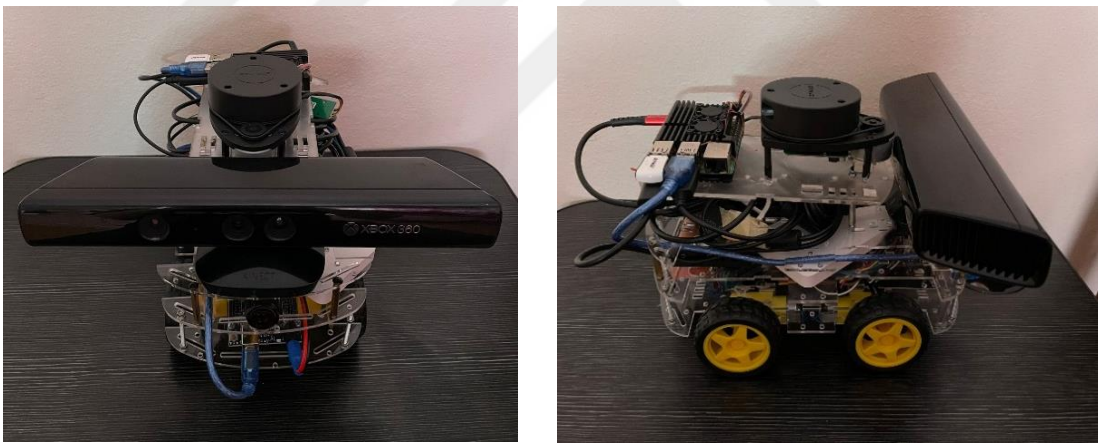


Figure 3.1. Front and Side Views of the Robot Platform

3.1. System Specifications

In the project, Raspberry Pi as a microcomputer and Arduino as a microcontroller was selected. It is aimed to establish a connection between them by serial communication via a USB cable.

Table 3.1. Robot Platform Specifications

Module	Model
Microcomputer	Raspberry Pi 4 Model B
Microcontroller	Arduino Uno
LiDAR	SLAMTEC RPlidar A1M8
Camera	Microsoft Kinect v1
Chassis	Four wheel chassis
Motor	DC gear motors
Motor driver	L298N motor driver

The overall design includes four-wheel mobile robot with an RPlidar and a Kinect camera connected to a Raspberry Pi 4 computer. The mobile robot has the following designed architecture shown in Figure 3.2.

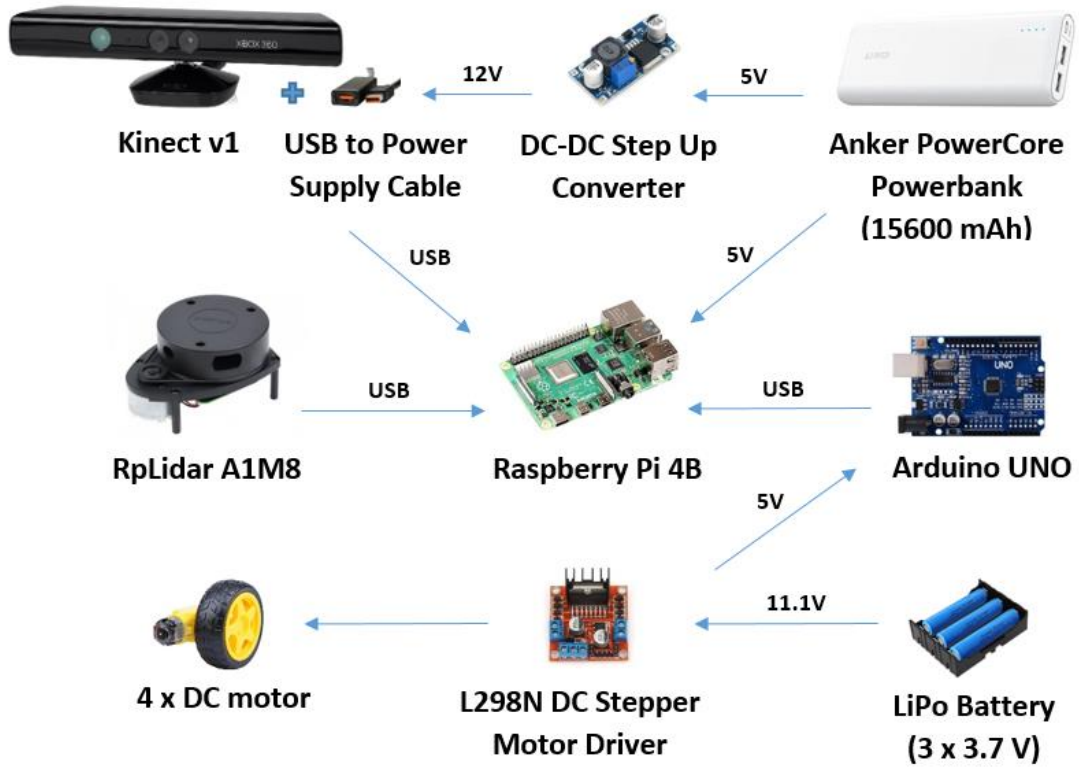


Figure 3.2. Hardware System Design

3.1.2. Microcomputer (Raspberry Pi)

Raspberry Pi 4 Model B features a 64-bit quad core having ARM-Cortex A72 processor at 1.5 GHz and has Wi-Fi, Bluetooth, Ethernet Port, USB 2.0, and USB 3.0 ports with 2GB, 4GB, and 8GB RAM options as displayed in Figure 3.3. In the project, option of 4 GB RAM was used as a microcomputer since it has more RAM than Raspberry Pi 3B+ which has 1 GB RAM. It operates with 5V minimum with 3A requirement which is powered via a USB-C port.

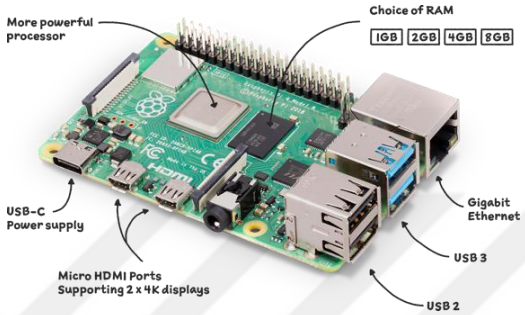


Figure 3.3. Components of Raspberry Pi[1]

3.1.3. Microcontroller (Arduino UNO)

Arduino UNO is a ATmega328P microchip based microcontroller which equipped with digital input/output pins (0-13), 6 of them provide PWM outputs (3, 5, 6, 9, and 11), 6 analog inputs pins (A0-A5) are represented in Figure 3.4. Power requirement to feed the board is between 7 and 20 volts.

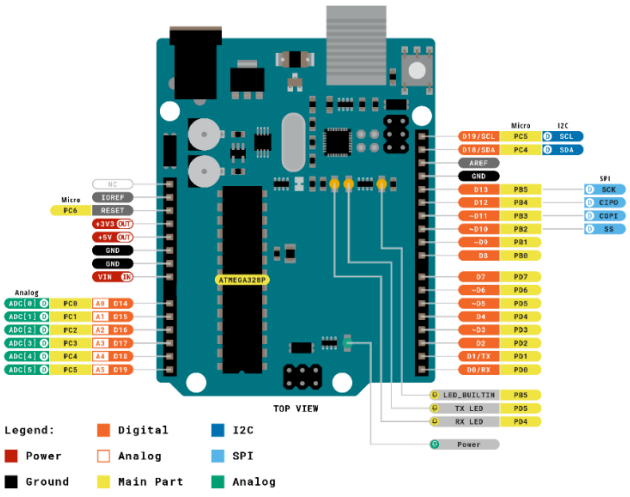


Figure 3.4. Pinout of Arduino Uno[2]

[1] https://www.raspberrypi.com/products/raspberry-pi-4-model-b/
[2] https://docs.arduino.cc/hardware/uno-rev3

3.1.4. Laser Scanner (RPlidar A1M8)

The SLAMTEC RPlidar A1 is a 2D laser scanner based on a laser triangulation approach to measure the range which has 360-degree angular and 12-meter distance range. The laser range scanner is attached to a motor through a belt and rotates in a clockwise direction continuously which is presented in Figure 3.5. While operating with a 5.5 Hz scanning frequency, 1450 points are sampled each turn also scan rate can be configured from 2 to 10 Hz according to motor speed. The RPlidar A1 was preferred in this project because it is low cost and has superior accuracy compared to Kinetic and ZED stereo cameras, a 360-degree field of view, and very low power requirement.

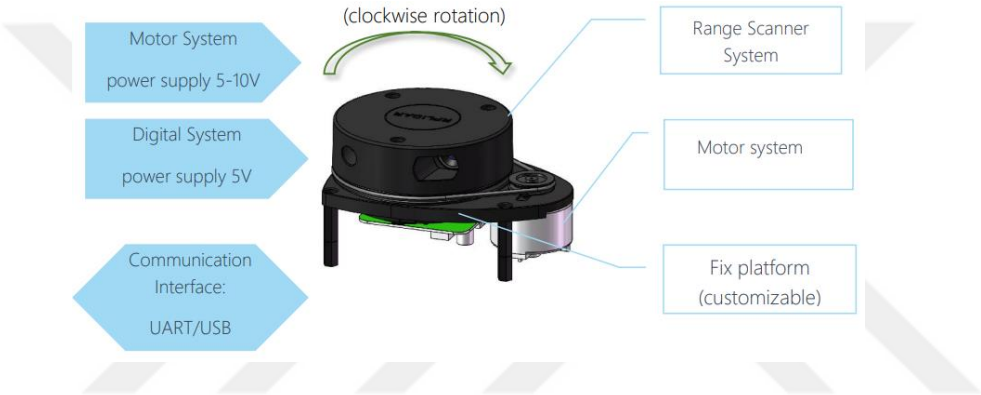


Figure 3.5. RPlidar A1 System Configuration (Shanghai Slamtec Co., 2009)

The RPlidar operates based on laser triangulation, emitting a modulated infrared laser signal which gets reflected from the object it detects as represented in Figure 3.6. It captures the reflected signal with the support of the vision acquisition system. Subsequently, the embedded digital signal processor in the RPlidar A1 processes the acquired data samples and constructs the corresponding distance and angle values of the detected object through a communication interface.

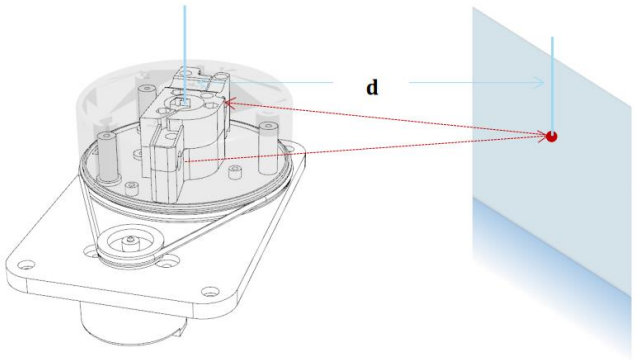


Figure 3.6. RPlidar A1 Working Principle (Shanghai Slamtec Co., 2009)

3.1.5. RGB-D Camera (Kinect V1)

The Microsoft Xbox Kinect camera V1.0 is based on a structured light approach that involves an infrared (IR) projector, an infrared camera, and an RGB camera as illustrated in Figure 3.7. Depth measurement is occurred by an IR projector and camera. Structured infrared light is emitted from the IR projector as tiny dots in large numbers over the area, then the IR camera captures the infrared dots that are arranged in a pattern of pseudo-random. Since this random pattern is generated systematically, the sensor can recognize the specific pattern. The sensor-to-point distance can be abstracted by a comparison between the captured image from the camera and the recognized displayed pattern (Miles, 2012). The range of depth measurement is between 0.8m to 4m.

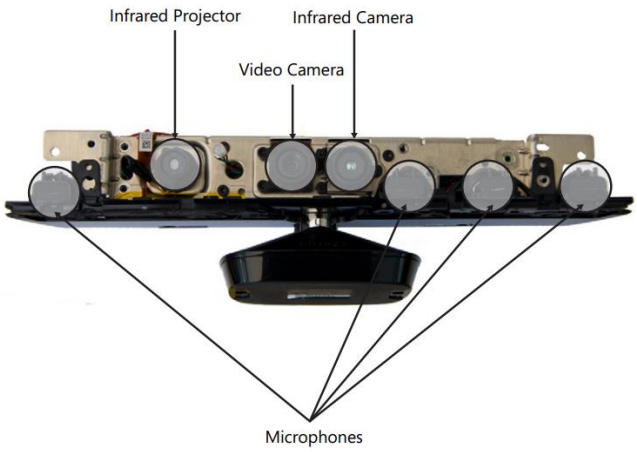


Figure 3.7. Kinect Camera Components (Miles, 2012)

The connector of the Kinect cable is designed for directly plugging to the Xbox therefore a USB/AC power supply cable was required that splits this connector into two cables. One of them is a USB-A cable for data transfer and the other is an AC adapter for the power supply. The Kinect V1 camera needs a 12V 1.5A power supply, thus the AC transformer plug was replaced with a step-up converter increasing the voltage from USB 5V to 12V, DC to DC.

3.1.6. Chassis

A 4-wheel drive robot car chassis was preferred in this project due to its low cost compared with omnidirectional wheels. The platform in Figure 3.8 has a length of 25.5 cm and a width of 15 cm. Rubber wheels with a diameter of 6.5 cm are attached to the end of the DC motors with 1:48 gear.

Figure 3.8. Robot Chassis with Motors and Wheels

3.1.7. Motor driver (L298N)

The module involves a L298N motor driver with a double H bridge and a 5V regulator which can drive 2 or 4 motors. The pins that control the speed of motors A and B are ENA and ENB pins while the pins are IN1, IN2, IN3, and IN4 for controlling direction. If there are jumpers on pins of ENA and ENB, the motor works at maximum speed. It is possible to control the speed of DC motors by adjusting the input voltage to the motors using pulse width modulation (PWM) signals after removing jumpers. The rotary direction can be controlled by inversing the current direction which is flown through the motor using the H-bridge.

3.1.8. Power Supplies

One cell of LiPo battery has a nominal voltage of 3.7V and a storage capacity of 2200 mAh. The battery pack contains 3 cells which means 11.1V output voltage. The capacity of the battery pack is calculated by multiplying the storage capacity with the battery voltage that is 24.42 Wh.

3.2. Drive Motor Architecture

On the motor driver module, the positive terminal of the battery connects with the 12V pin, and the negative terminal of the battery and the GND pin of the Arduino is connected to the GND pin as seen in Figure 3.9. Motor driver’s input and enable pins of the (ENA, IN1, IN2, IN3, IN4, and ENB) are connected to (10, 9, 8, 7, 6, and 5) respectively. Note that, Arduino output pins (10 and 5) which are connected to Enable pins (ENA and ENB), are PWM-enabled.

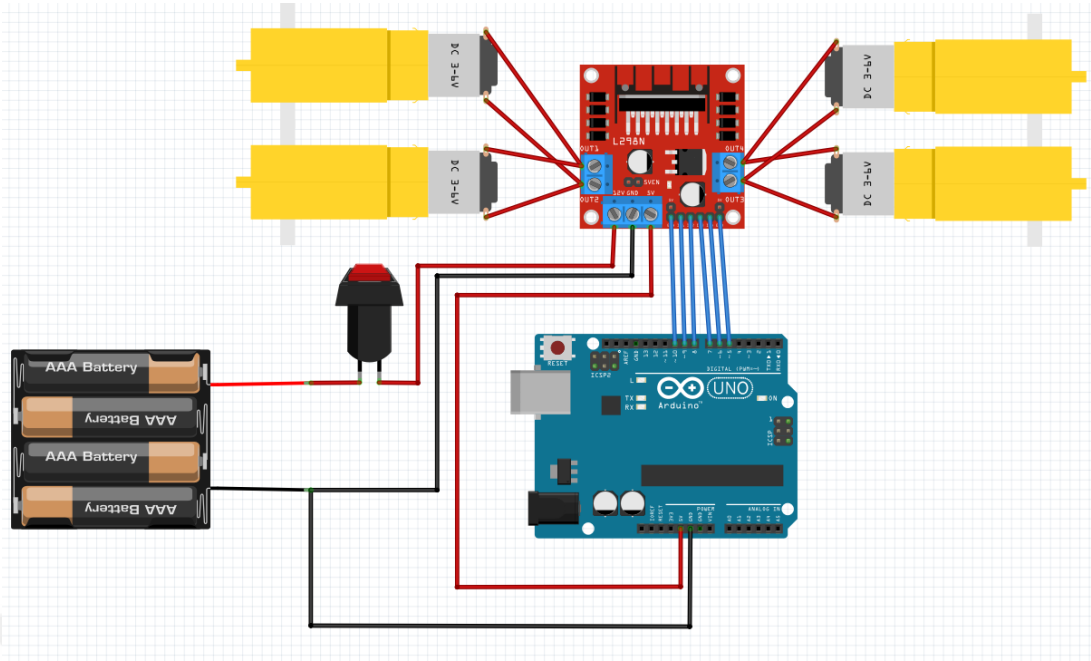


Figure 3.9. Drive Motor Connection Schematic

3.3. Kinematic Model

The kinematic model is required for understanding and predicting the motion and behavior of the robot platform. The kinematic model represented in Figure 3.10 is based on Houssein et al. (2019). The front and back wheels of the mobile robot are connected to each other on both sides which forms a differential drive.

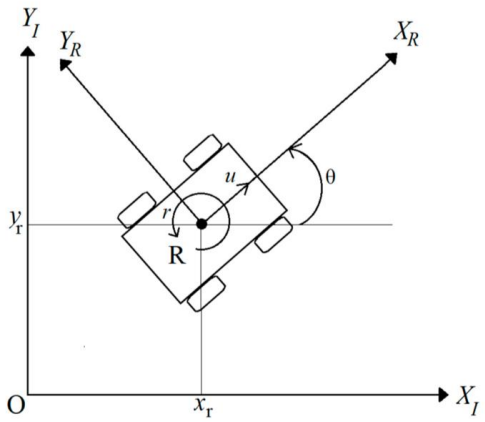


Figure 3.10. Mobile Platform Kinematics (Houssein et al., 2019)

The inertial frame is represented by the global reference O: $\{X_I, Y_I\}$. The center of the mobile platform is denoted as R, and its position is defined within the inertial frame

$\{X_I, Y_I\}$. The heading angle θ, measures the difference between the mobile robot's attached frame $\{X_R, Y_R\}$ and the inertial frame $\{X_I, Y_I\}$ indicating the deviation.

The following vector represents the position of the mobile platform in the inertial frame:

$$P_R = \begin{bmatrix} x_r \\ y_r \\ \theta \end{bmatrix} \tag{19}$$

consisting of x_r and y_r are coordinates that denote the planar position of the robot in relation to the global reference frame $\{X_I, Y_I\}$. Additionally, the vector includes the heading angle θ, which indicates the direction in which the robot is oriented.

Hence, the model of the system is represented by

$$\begin{bmatrix} \acute{x} \\ \acute{y} \\ \acute{\theta} \end{bmatrix} = \begin{bmatrix} cos\theta & 0 \\ sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ r \end{bmatrix} \tag{20}$$

incorporates the linear velocities $\acute{x}$ and $\acute{y}$ in the robot's frame $\{X_R, Y_R\}$, as well as the angular velocity $\acute{\theta}$. The control inputs for the linear and angular velocity are denoted as u and r, respectively, influencing the robot's motion and behavior.

The robot is provided with a waypoint $\omega(X_\omega, Y_\omega)$ in its inertial coordinates as the target for navigation. As a local planner, it assumes the existence of at least one achievable path connecting the position of the platform $P(x_r, y_r)$ to the end position $\omega(X_\omega, Y_\omega)$.

Navigating the mobile robot to the desired goal successfully in an environment without obstacles is the initial objective. To measure the separation between the robot's current position (p_R) and the destination (p_ω), the Euclidean distance d is denoted as:

$$d = \sqrt{(x_\omega - x_r)^2 + (y_\omega - y_r)^2}. \tag{21}$$

The angle α represents the direction in which the robot is oriented towards the waypoint $\omega(X_\omega, Y_\omega)$

$$\alpha = \beta - \theta \tag{22}$$

where β can be expressed as

$$\beta = atan2(y_\omega - y_r, x_\omega - x_r). \tag{23}$$

Fulfillment of the following assumption is necessary for confirming that the robot has reached the destination:

$$\lim_{t\to\infty} d(t) = d_{tolerance} \lim_{t\to\infty} \alpha(t) = 0 \tag{24}$$

where predefined distance $d_{tolerance}$ serves as a threshold indicating that the robot has successfully reached its waypoint. Consequently, this controller guarantees the mobile robot's arrival at a designated waypoint within an environment without obstacles:

$$\begin{aligned} u &= k_u d^2 \\ r &= k_r \propto, \end{aligned} \tag{25}$$

where $k_u$and k_r represent the positive gains.

CHAPTER 4
IMPLEMENTATION

This chapter emphasizes the implementation aspects of the thesis, specifically examining the effective utilization of different packages and tools within the ROS framework for the successful navigation of the platform. The importance of each package and its contribution towards accomplishing the goals outlined in the thesis is explained. During the implementation phase, several elements which are sensor readings, SLAM packages, localization technique, path planning, and visualization tools are integrated to enhance the robot's navigation abilities. Within the scope of this thesis, several ROS packages preferred for certain purposes are presented in Table 4.1.

Table 4.1. Main Packages of the Implementation

Package name	Task
rplidar_ros	Provides scan data (/scan) from the laser
freenect	Provides RGB-D data from the Kinect camera
TF	Tracks several coordinate frames over time
rtabmap_slam	Performs RGB-D SLAM
hector_mapping	Provides LiDAR-based odometry
move_base	Plans and implements action toward the goal
serial_python	Serial connection to Arduino
RVIZ	Visualization tool

The following explanations about descriptions of these packages have been provided. The “rplidar” package allows users to obtain data such as raw point cloud data and distance measurements from the RPlidar sensor. The “Freenect” package enables communication with the Kinect sensor by transmitting data streams like color images and depth maps. The transforms (tf) tree is used for defining the position and

orientation of sensors related to the robot base. The “rtabmap_slam” package is a SLAM library that utilizes visual and depth data to perform mapping and localization. The hector_mapping is another SLAM library for building a 2D occupancy grid map of its surroundings while simultaneously estimating its pose using sensor data. The “move_base” is obligated for path planning and navigation by integrating the TEB local planner with the ROS global planner to create a safe route to the destination. The “serial_python” package sets a serial communication between the Raspberry Pi computer and the Arduino board to send velocity commands for motor control.

Integration between the “move_base” package and RTAB-Map (MathieuLabbe, 2023a) seen in Figure 4.1 allows navigation and mapping functionalities. The 2D grid map generated by RTAB-Map serves as input to the “move_base”. It provides essential environment information to the global planner, enabling path planning on a known map. The map information is then utilized by the “move_base” for re-planning and obstacle avoidance. In addition, RTAB-Map offers localization within the map. This localization data enhances the “move_base” package's ability to generate optimal paths and accurately navigate the robot within the known environment.

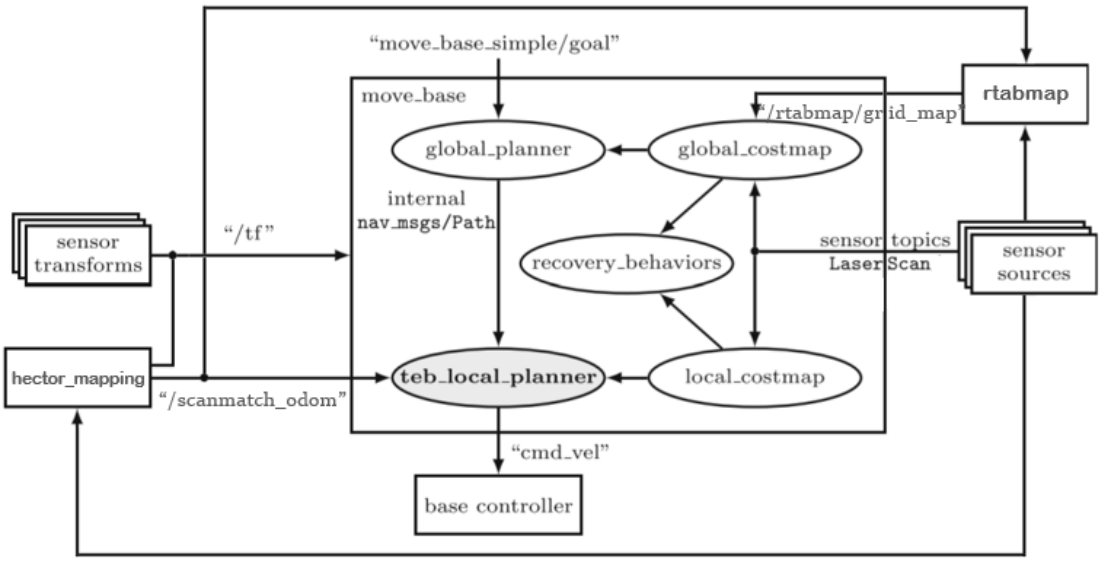


Figure 4.1. Software Architecture of the Navigation Process

4.1. Transform Configurations

Static links from the “/camera_link” and “/laser” frames to the “/base_link” frame, and “/base_link” frame to the “/base_footprint” frame were made using the "static_transform_publisher" package as given in Figure 4.2.

```
   <node pkg="tf" type="static_transform_publisher"
name="camera" args="0.1 0 0.05 0 0 0 base_link
camera_link 100" />
   <node pkg="tf" type="static_transform_publisher"
name="laser" args="0 0 0.09 0 0 0 base_link laser
100" />
   <node pkg="tf" type="static_transform_publisher"
name="base_footprint_to_base_link"
args="0.0 0.0 0.11 0.0 0.0 0.0 /base_footprint
/base_link 100" />
```

Figure 4.2. Static Transform Publisher Launch Code

The “base_link” frame serves as the parent frame, while the “camera_link” and “laser” frames serve as the child frame whereas the transform moves from parent to child. The center point of the Kinect is 0.1 meters forward (x-axis) and 0.05 above (z-axis) the center point of the platform. Thus, the transformation is defined from the “laser” to the “base_link” as (x: 0.1m, y: 0.0m, z: 0.05m). If the same logic applies, the center of the laser scanner is located 0.09 meters above the base_link (x: 0.0m, y: 0.0m, z: 0.09m), and the base_link frame is located 0.11 meters above the base_footprint, which is defined as the center of robot's base (x: 0.0m, y: 0.0m, z: 0.11m) (JunyaHayashi, 2021). The established transformation is given in Figure 4.3 via RVIZ.

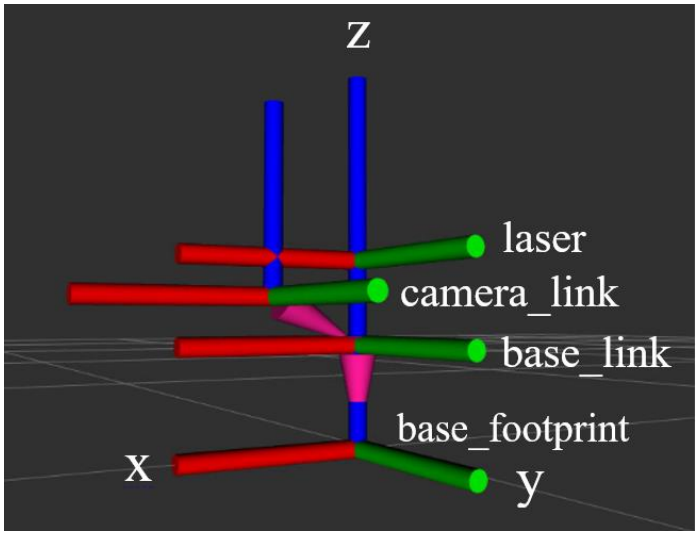


Figure 4.3. Coordinate Frame Transformation

In addition to all static transformations, hector_mapping produces the link from /base_footprint frame to /hector_map frame, and /rtabmap generates the link from /hector_mapping frame to /map frame. The transform tree created within the scope of the thesis is shown in Figure 4.4.

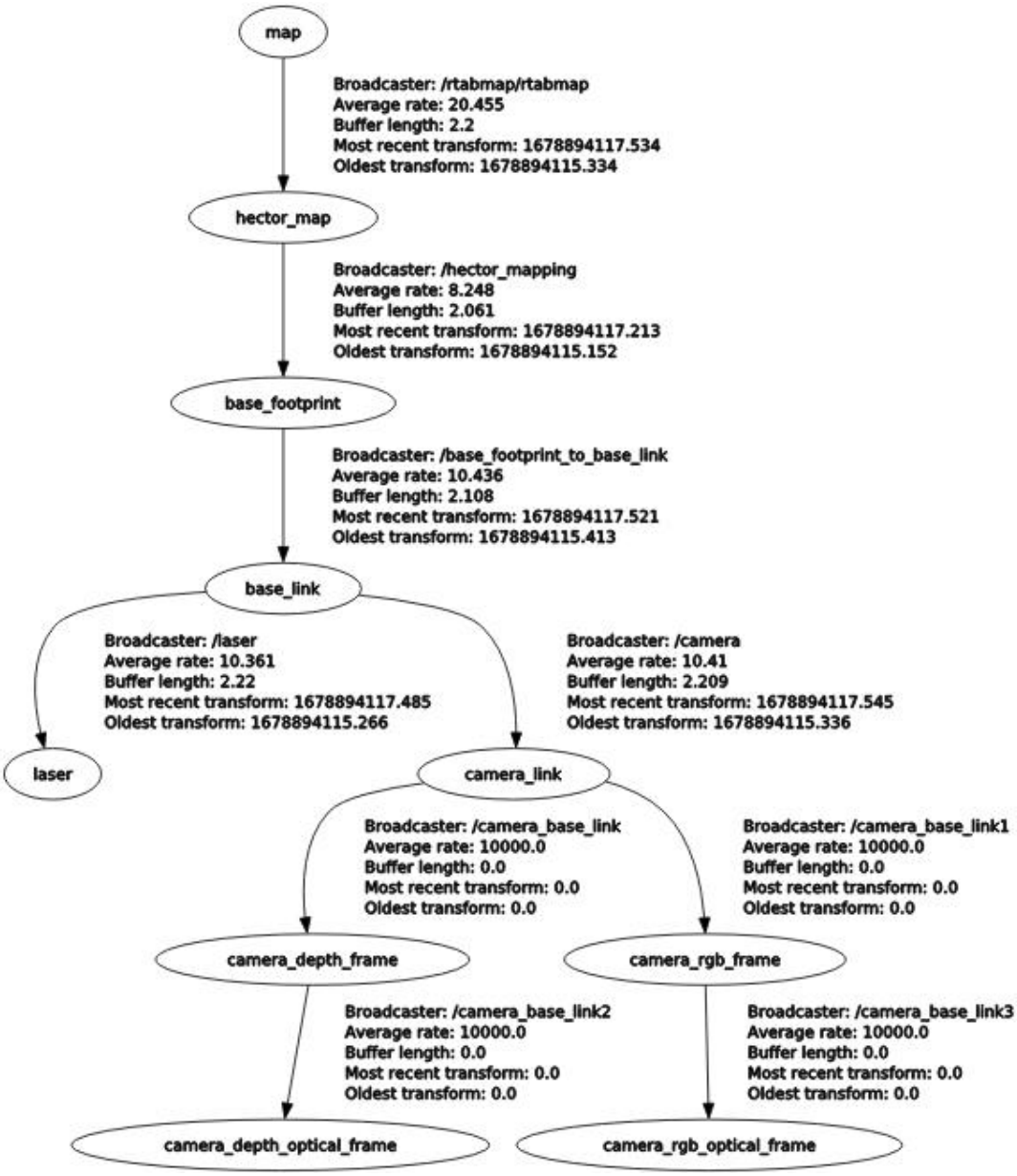


Figure 4.4. Transform Tree of the System

4.2. Mapping the Environment

Once the sensors publish their data to the relevant topics, the hector mapping parameters are configured. These parameters enable the estimation of odometry through scan matching when both the "pub_map_scanmatch_transform" and "pub_odometry parameters" are enabled (StefanKohlbrecher, 2011). The odometry

information contained in the published “/scanmatch_odom” topic from hector mapping is then subscribed to by the “/rtabmap” node.

The "/scan" topic that has a "sensor_msgs/LaserScan" type message contains the data obtained by the laser sensor published by the "rplidarNode" node. Then, the “/hector_mapping” node subscribes to this "/scan" topic. Furthermore, it receives updates from the "/tf" node the "tf/tfMessage" type, which contains the relations of the nodes with each other which is required for the operation of the "hector_mapping" node. The "/tf" node has the transform information of the laser scanner (/laser) position on the robot relative to the robot’s center (/base_link). These transformations are provided statically using the "static_transform_publisher" node.

A nodelet called "rgbd_sync" is used to synchronize and process RGB-D data, consisting of RGB images and depth images. The configuration sets up topic remapping and parameter settings as Figure 4.5, enabling efficient data transmission and improved performance within the ROS system. The subscribed topics of the "rgbd_sync" nodelet are "rgb/image", "depth/image", and "rgb/camera_info" are synchronized based on their timestamps, and the “rgbd_image” topic is published as output.

```
  <node pkg="nodelet" type="nodelet" name="rgbd_sync"
args="standalone rtabmap_sync/rgbd_sync"
output="screen">

      <remap from="rgb/image"
to="/camera/rgb/image_rect_color"/>

      <remap from="depth/image"
to="/camera/depth_registered/image_raw"/>

      <remap from="rgb/camera_info"
to="/camera/rgb/camera_info"/>

      <remap from="rgbd_image" to="rgbd_image"/>
<!-- output -->

      <!-- Should be true for not synchronized camera
topics (e.g., false for kinectv2, zed, realsense,
true for xtion, kinect360)-->

      <param name="approx_sync"    type="bool"
value="true"/>

  </node>
```

Figure 4.5. RGB-D Synchronization Node (MathieuLabbe, 2023b)

The “rqt_graph” tool provides a visual representation of the previous code sample in Figure 4.6 which publishes the “rgbd_image” node that contains synchronized RGB and depth images captured by the camera sensor.

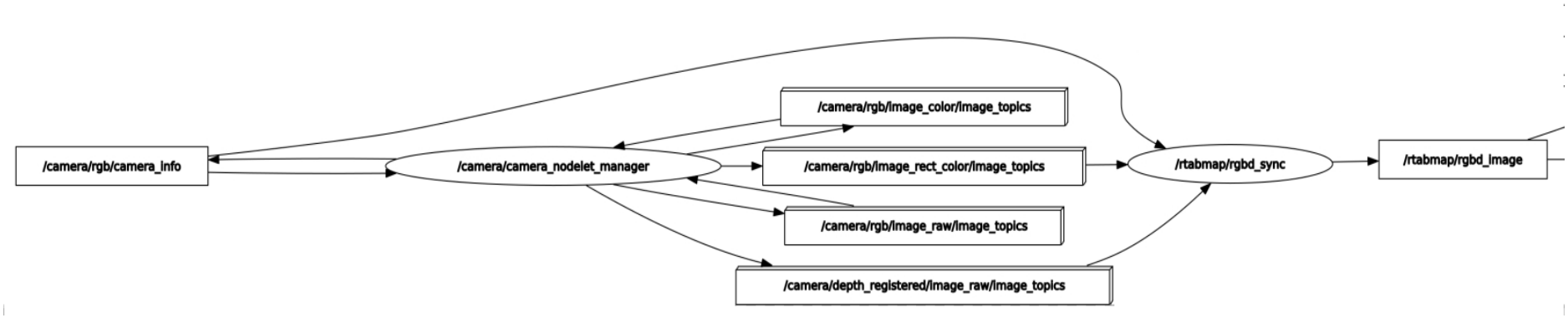


Figure 4.6. RQT Graph of the “rgbd_sync”

The configuration of a ROS node named "rtabmap" from the "rtabmap_slam" package includes parameter settings and arguments to customize the behavior of the node. In Figure 4.7, the path to the database file used by the "rtabmap" node is defined as "rtabmap.db". Additionally, if the “localization:=false” argument is provided when the node is launched, the "--delete_db_on_start" argument ensures that the database is cleared by deleting the existing map when starting the mapping section.

```
  <arg name="database_path"   default="rtabmap.db"/>
  <arg     if="$(arg localization)" name="args"
default=""/>
  <arg unless="$(arg localization)" name="args"
default="--delete_db_on_start"/>
```

Figure 4.7. Configuration Parameters for RTAB-Map Database and Localization

Figure 4.8 contains parameters related to the localization mode. If the "localization" argument is set, the "Mem/IncrementalMemory" is “false” which indicates that the system should not use incremental memory update in the localization mode. Otherwise, it allows the system to update the map incrementally with new data as it explores the environment in mapping mode. Furthermore, the "Mem/InitWMWithAllNodes" parameter is for initializing the Working Memory (WM) with all nodes to store information about the environment based on the localization mode.

```
  <!-- localization mode -->
  <param     if="$(arg localization)"
name="Mem/IncrementalMemory" type="string"
value="false"/>
  <param unless="$(arg localization)"
name="Mem/IncrementalMemory" type="string"
value="true"/>
  <param name="Mem/InitWMWithAllNodes" type="string"
value="$(arg localization)"/>
```

Figure 4.8. Configuration Parameters for Localization Mode

Basic descriptions of the parameters related to the configuration of the RTAB-Map system are given (MathieuLabbe, 2023b): “Reg/Strategy” determines the registration strategy “0” for visual registration, “1” for ICP (Iterative Closest Point) registration, and “2” for a combination of visual and ICP registration. By setting the “Reg/Strategy” parameter to 1, the ICP algorithm using laser data was selected to improve the precision of global loop closures detection. “Vis/MaxDepth” specifies the maximum

depth allowed for visual feature extraction, with 8.0 as the value used. “Vis/InlierDistance” sets the correspondence distance threshold for 3D visual words. “Optimizer/Slam2D” with value “true” controls whether SLAM optimization is performed in 2D. “Reg/Force3DoF” with value “true” enforces a 3 degrees of freedom (DoF) constraint during registration. “GridGlobal/MinSize” determines the minimum size of grid cells in the global map. “RGBD/ProximityBySpace” enables proximity detection of RGB-D sensor data based on spatial distances. “Icp/CorrespondenceRatio” sets the ratio of correspondences used in the ICP registration algorithm. “Icp/VoxelSize” specifies the voxel size used in the ICP algorithm for down-sampling point clouds.

4.3. Navigating the Platform

The navigation stack configured with the TEB local planner comprises several components and configurations that enable autonomous navigation within the environment. The TEB local planner is one of the planners available in the navigation stack, which generates dynamically feasible trajectories for the robot to follow. The choices of plugins and relevant parameter values for the global and local planners are represented in Figure 4.9.

```
  <param name="base_global_planner"
value="global_planner/GlobalPlanner" />
  <param name="planner_frequency" value="1.0" />
  <param name="planner_patience" value="5.0" />
  <param name="base_local_planner"
value="teb_local_planner/TebLocalPlannerROS" />
  <param name="controller_frequency" value="5.0" />
  <param name="controller_patience" value="15.0" />
```

Figure 4.9. Configuration Parameters for Planners

The “move_base” receives the map from “rtabmap”, scan data, and odometry from “hector_mapping” and outputs the command velocity (/cmd_vel) for the motor as seen in Figure 4.10. The following section describes the parameters in the configuration files with the same name as the headers.

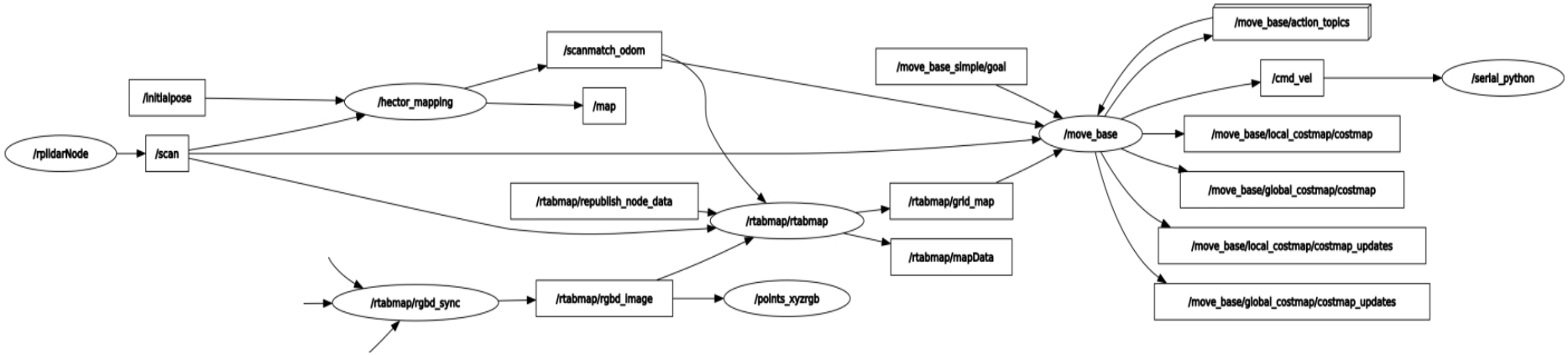


Figure 4.10. RQT Graph of the Navigation Process (continues from /rtabmap/rgbd_sync)

4.3.1. Costmap Common Parameters

The configuration file named “costmap_common_params.yaml” contains parameters related to the costmap used for navigation which defines how obstacles and inflation are handled. The important parameters are listed with their explanations below (NickLamprianidis, 2018):

- The “robot_radius” with a radius of 0.17 meters defines the size of the robot's footprint. Robot maintains a safe distance from the inflated obstacles in the costmap based on this parameter.
- The “footprint_padding” adds extra distance between the robot's footprint and the obstacle to avoid collisions.
- The “map_type: costmap” specifies the type of map used, which in this case is a costmap.
- The “obstacle_layer” is responsible for handling obstacle information in the costmap. The obstacle is detected in the range set by “obstacle_range: 3.0” in meters. The radius of inflated obstacles set by “inflation_radius” is 0.2 meters which inflate by assigning higher costs to the surrounding cells.
- The “observation_sources” defines the laser scanner as the observation source to detect obstacles for the costmap.
- The “inflation_layer” creates a safety margin by inflating the obstacles in the costmap. “inflation_radius” is set to 0.5 which specifies the maximum distance from an obstacle where costs are taken into account.
- The “static_layer” uses the static map as a reference for navigation which specifies the published topic as "/map".

4.3.2. Costmap Converter Parameters

The costmap conversion functionality is configured to combine point obstacles into clusters which improves the local planner efficiency with the TEB planner configuration (Zacwitte, 2016).

- The "costmap_converter_plugin" determines the plugin assigned to handle the conversion of the costmap.

- The " CostmapToLinesDBSRANSAC" utilizes the RANSAC (Random Sample Consensus) algorithm to convert the costmap into lines. Configuring parameters for the conversion process include cluster distance thresholds, minimum required points for cluster formation, distances for inlier classification in RANSAC line fitting, iteration limits, and outlier handling options.

4.3.3. Global Costmap Parameters

The following parameters focus on the global cost map configuration in "move_base" (Zheng, 2021):

- The “global_frame” defines the coordinate frame configured as a "map" which is used by the global costmap. Comprehensive depictions of obstacles and free spaces on a global scale are represented within this frame.
- The “robot_base_frame” configured as "base_link" defines the coordinate frame which is utilized to adjust the global costmap to correspond with the robot's location and direction.
- The “static map: true” determines if a pre-existing map of the environment is employed as the base layer for the global costmap which can serve as a navigation reference.
- The “plugins” defines the plugins utilized in the global costmap for processing and incorporating data into the costmap. Three plugins are employed: “static_layer” plugin manages the static map data by integrating into the global costmap, offering a fixed representation of the environment. “obstacle_layer” plugin handles obstacle data within the global costmap by detecting and updating obstacles using sensor data. “inflation_layer” plugin expands the obstacles present in the global costmap by creating a safety buffer around inflated obstacles.

4.3.4. Local Costmap Parameters

The parameters for the configuration of the local costmap are given below (Zheng, 2021):

- The “global_frame” determines the coordinate frame used by the local costmap which aligns with the global map frame by setting it to “map”.
- The “robot_base_frame” defines the coordinate frame associated with the “base_link” to match the local costmap with the position and orientation of the robot.
- The “static map: false” implies that the static map is not used as a base layer for the local map since it is updated according to sensor data where the robot is located.
- The “rolling_window: true” activates the rolling window mode for the local costmap which is dynamically centered around the robot's position, providing a localized view of the environment.
- The “plugins” in the local costmap utilize by defining their respective types. The “static_layer” plugin represents the static map data within the local costmap. The “obstacle_layer” plugin spots and updates obstacles within the local costmap according to sensor data from the robot’s surrounding which maintains an updated representation of the environment.

4.3.5. TEB Local Planner Parameters

The following parameters are used to configure the TEB local planner (ChristophRoesmann, 2020):

- The “Trajectory Parameters” determine the characteristics of the planned trajectory, such as time intervals, maximum samples, and planning considerations. They ensure the generation of smooth and feasible trajectories while accounting for the robot's dynamics.
- The “Robot Parameters” define the maximum velocities, accelerations, and turning radius of the robot. By configuring these values, the planner can generate trajectories that comply with the physical constraints of the robot.
- The “Footprint Model” parameters represent the shape and size of the robot within the planner. In this case, a "point" footprint model is used, simplifying the robot's shape to a single point. This assumption allows the planner to account for the robot's ability to rotate in place.

- The “Goal Tolerance” parameters determine the acceptable proximity to the goal position and orientation. The chosen values specify how close the robot must be to consider the goal as reached.
- The “Obstacles” section includes parameters related to handling obstacles. It defines the minimum distance to maintain from obstacles, inflation distance for cost calculation, and considerations for static and dynamic obstacles. These settings ensure that the planner generates trajectories that avoid collisions with obstacles in the environment.
- The “Costmap Converter” parameters define the plugin responsible for converting the costmap representation of obstacle information. The spin thread and rate settings ensure efficient conversion and integration of the costmap data into the planner.
- The “Optimization” parameters control the trajectory optimization process. They specify the number of iterations, weight factors for different cost terms (e.g., velocity, acceleration, obstacles), and other factors that influence the optimization process.
- The “Homotopy Class Planner” section pertains to homotopy class planning, which explores different feasible paths based on the environment's topological properties. The settings determine the number of classes, cost factors, and various aspects of the homotopy class planning algorithm.
- The “Recovery” parameters define recovery behaviors in case of failures or obstacles. They include settings for horizon shrinking, oscillation detection and recovery, and durations related to ensuring the planner can recover from challenging situations.

4.4. Controlling the Platform

An Arduino-based robot needs a ROS node to control its movement which operates using the “geometry_msgs/Twist” message to receive commands for linear and angular velocities via the "cmd_vel" topic. The code includes the necessary header files, namely “ros.h” and “geometry_msgs/Twist.h”, to establish communication with the ROS environment and handle the Twist message for velocity commands.

Two variables, x, and z, are defined within the code. They serve the purpose of storing linear and angular velocities received in the Twist message. A ROS node handle (nh) is created, and a subscriber is initialized to receive messages on the "cmd_vel" topic. The subscriber's callback function, “velCallback”, is triggered whenever a new Twist message is received.

The values given in Table 4.2 are sent by ROS using the “teleop_twist_keyboard” package which is used for setting direction commands from the keyboard. From this examination, the underlying logic of the Arduino code was derived.

Table 4.2. Velocity Command Values of the “teleop_twist_keyboard”.

		Forward (m/s)	**Backward (m/s)**	**Right (m/s)**	**Left (m/s)**
Linear	x	0.5	-0.5	0	0
	y	0	0	0	0
	z	0	0	0	0
Angular	x	0	0	0	0
	y	0	0	0	0
	z	0	0	-1.0	1.0

In the “velCallback” function, the code interprets the velocities from the Twist message to determine the robot's motion:

- If linear velocity (x) is greater than 0.1, the robot moves forward.
- If linear velocity (x) is less than -0.1, the robot moves backward.
- If angular velocity (z) is less than -0.1, the robot turns right.
- If angular velocity (z) is greater than 0.1, the robot turns left.
- If none of the above conditions are met, it stops the motors.

The code uses “digitalWrite” and “analogWrite” functions to control the motor driver accordingly for each motion scenario. In the loop function, the “nh.spinOnce()” is

called to process any incoming ROS messages, and then the code delays for 10 milliseconds before the next iteration.

CHAPTER 5
EXPERIMENT RESULTS

This section provides a detailed examination of the mapping procedure with RTAB-Map SLAM as well as observing navigation abilities, and obstacle performance of the TEB local planner in the ROS framework.

5.1. Constructed Map

The constructed map is presented in Figure 5.1 from two different perspectives, which displays the door, library, table, chair, and two beds within the room. Distinctly, rectangular voids shown on the map correspond to the tops of the beds located on the two sides, and the top of the obstacle at the center remains an unexplored territory due to its inaccessibility to the camera's field of view.

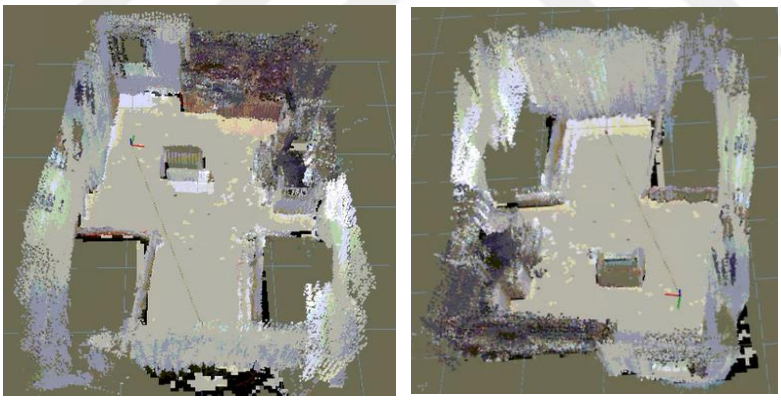
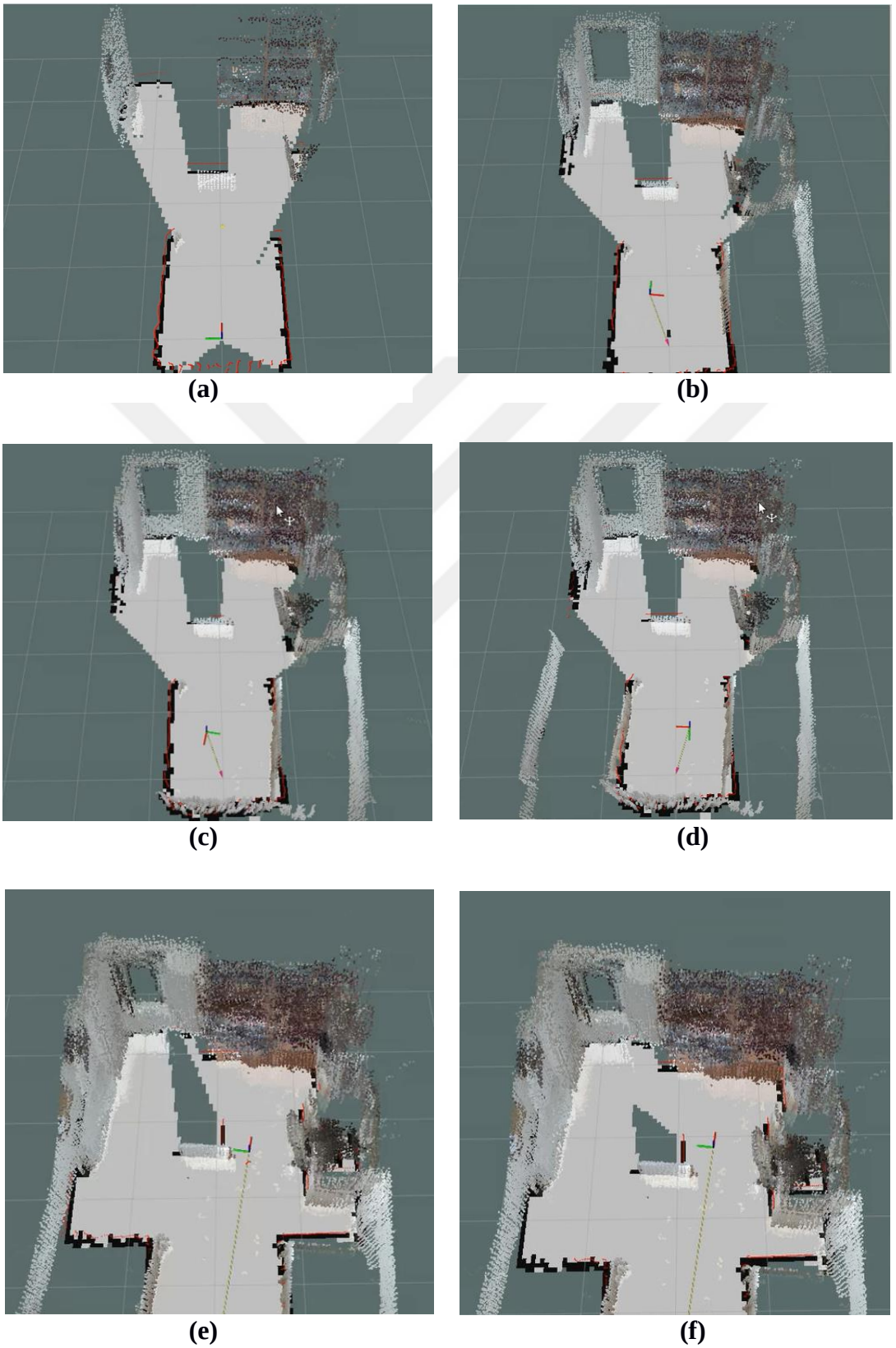


Figure 5.1. Different Perspectives of the Constructed Map

The step-by-step depiction of the mapping process is presented in Figure 5.2. The map is incrementally formed following the collection of data in the robot's forward direction. It is seen in Figure 5.2 (a) that the data collected from only one direction at the very beginning of the mapping process is processed on the map. As the robot turns to the right, the point clouds of the right walls in Figure 5.2 (b), the back walls in Figure 5.2 (c), and the left walls in Figure 5.2 (d) continue to be added to the map. At this stage, unexplored areas persist since the robot has not yet investigated the spaces concealed behind the obstacles. Notably, the unknown region diminishes as the robot

turns left and initiates the gathering of information regarding the previously unseen blind spot, as illustrated in Figure 5.2 (e) and Figure 5.2 (f). As the robot approaches the objects, the frequency of point clouds increases.



(a) **(b)**

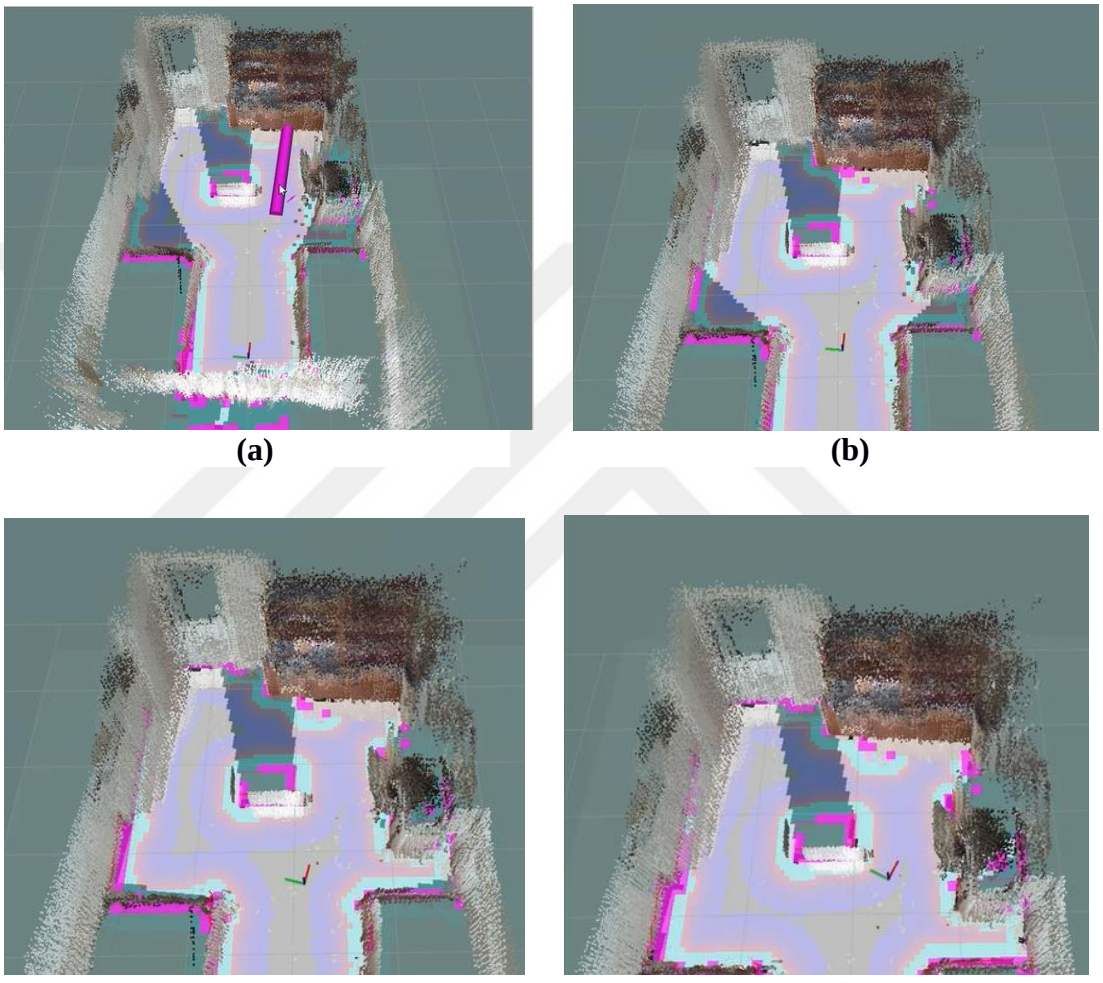
(c) **(d)**

(e) **(f)**

Figure 5.2. Mapping Process in RVIZ.

5.2. Navigation Tests

The navigation process of conducting the obstacle avoidance test in the right direction is given in Figure 5.3. The initial step is the selection of the target point using the purple arrow with the “2D Nav Goal” button in RVIZ as in Figure 5.3 (a). It is shown that the movement of the platform as in Figure 5.3 (b) changes its direction towards the right in Figure 5.3 (c), whereas its arrival at the intended position in Figure 5.3 (d).

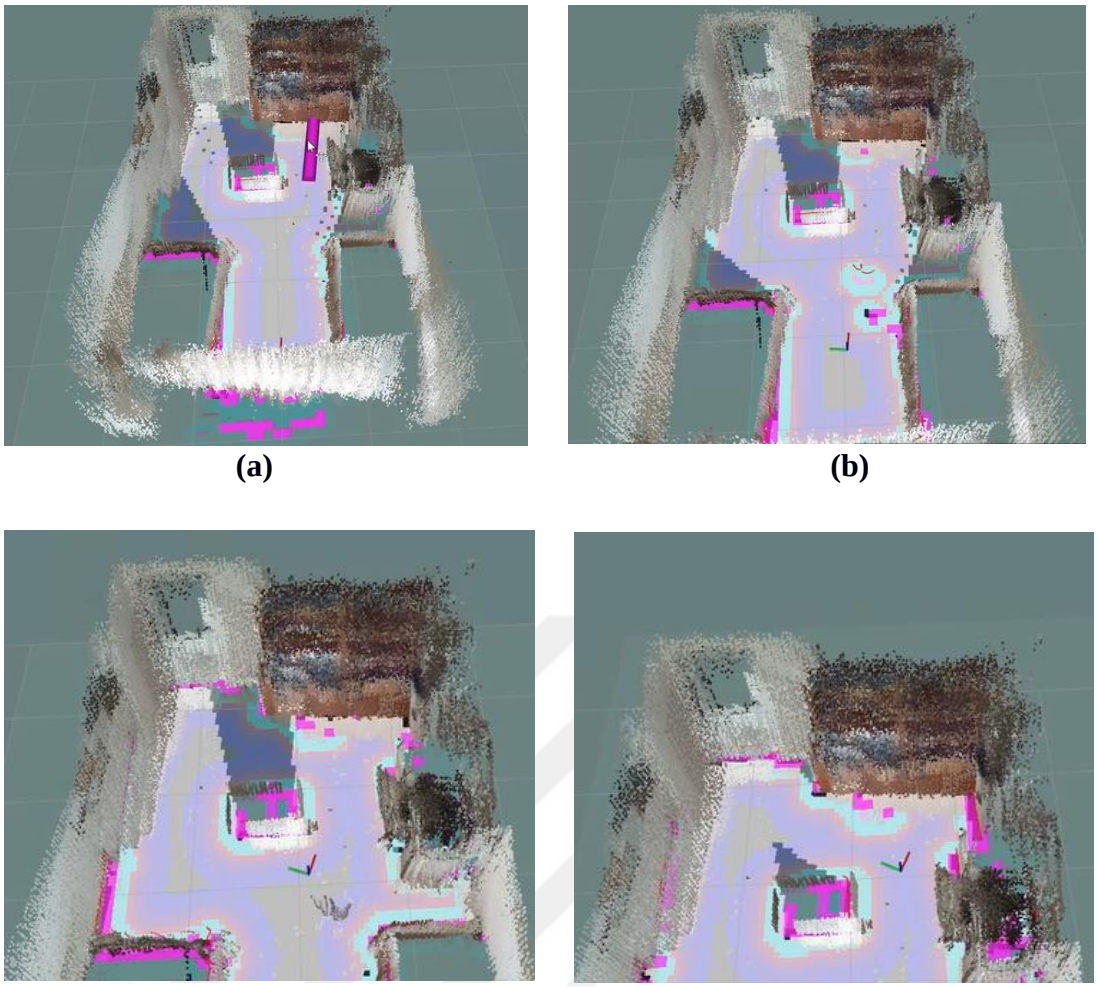


(a) **(b)**

(c) **(b)**

Figure 5.3. Testing a Right Turn.

The next experiment is to observe the performance of the TEB local planner in ROS when encountering dynamic objects. A goal location was set for the robot to reach within the environment in Figure 5.4 (a). While the robot was in motion as Figure 5.4 (b), an individual passed in front of it and waited a while. The TEB local planner adapts to the environmental change and updates the robot's trajectory in real-time. It adjusted the path dynamically in Figure 5.4 (c) and reached the goal by passing the obstacle in Figure 5.4 (d).



(a) **(b)**

(c) **(d)**

Figure 5.4. Testing Dynamic Obstacle Avoidance.

Testing left turn ability is similar to the right turn test, except the robot needs more space due to the position of the obstacle to complete its maneuver and pass the obstacle. In order to provide this, the “footprint_padding” parameter adds an extra buffer to the robot's footprint when generating the costmap. At the beginning a goal position is set by "2D Nav Goal" button as shown in Figure 5.4 (a). The diagram illustrates that the platform's movement, depicted in Figure 5.4 (b), results in a change of its direction towards the left in Figure 5.4 (c), ultimately leading to its arrival at the desired position in Figure 5.4 (d).

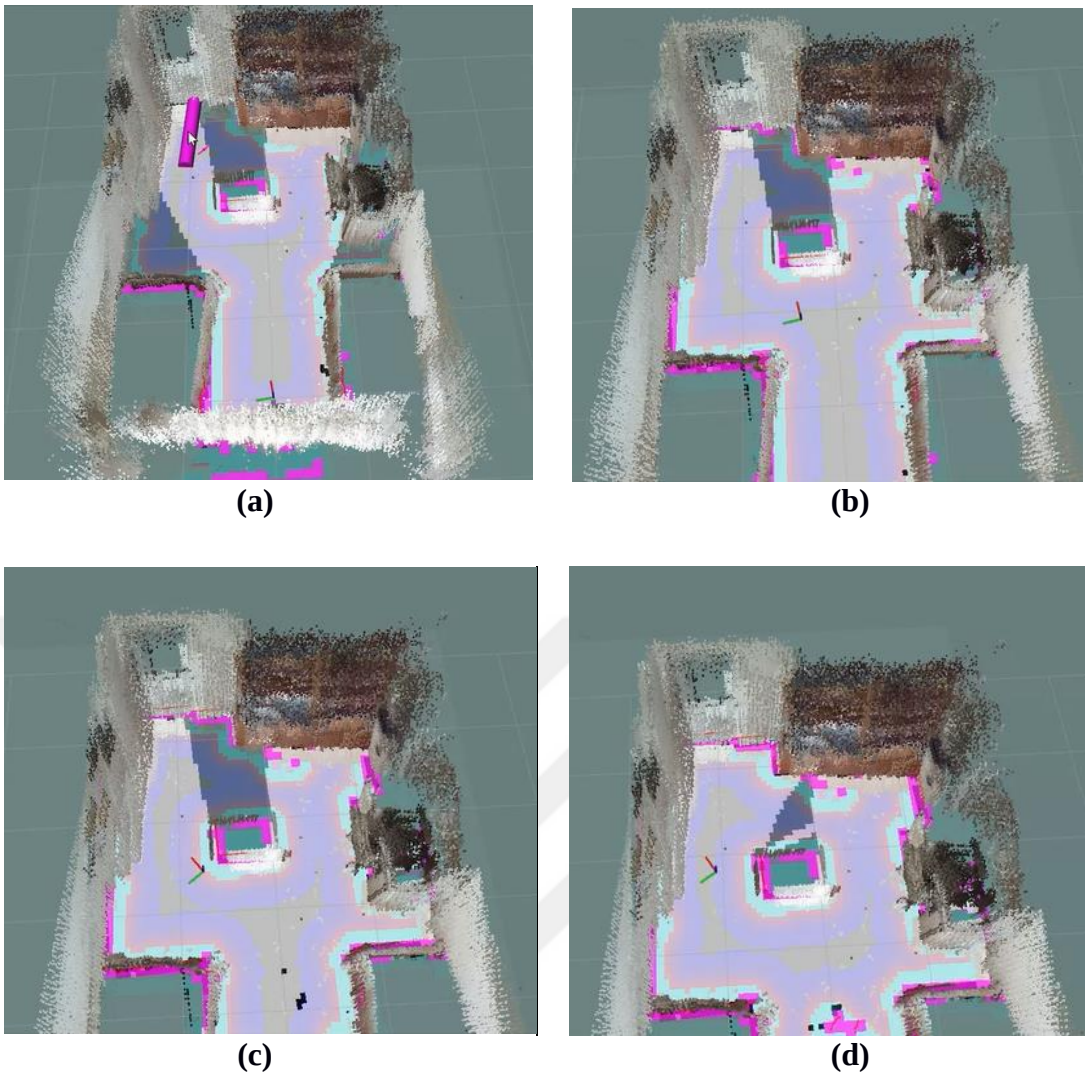


Figure 5.5. Testing a Left Turn.

Finally, the precision of reaching the target position was evaluated through the left turn test which involves sending a goal to the move_base action server. The robot's target position is set at the x-coordinate of 2 m, the y-coordinate of 1 m, and the w-component of the quaternion to 0 for orientation. This means that the orientation is aligned with the x-axis, resulting in the robot facing directly along that axis. The robot executed multiple maneuvers to achieve this objective; however, these actions were essential to validate its attained position. Table 5.1 displays the results of each test, including the actual goal coordinates (x, y), as well as the error distance between the robot's actual position and the intended goal. This error distance signifies the discrepancy between the robot's final position and the intended destination.

Table 5.1. Left Turn Test Results for the Goal Accuracy

Test No	Goal Coordinates (cm)		Error Distance (cm)
	x	y	
1	5	0.8	5.06
2	5.3	1.6	5.54
3	5.3	2	5.66
4	6.4	1.7	6.62
5	4.7	3	5.58
6	4.9	4.6	6.72
7	3	8.6	9.11
8	5.5	3.2	6.36
9	6.1	6.9	9.21
10	4.5	6.3	7.74

Based on the sequence of the test results, Figure 5.6 presents the representation of the destination points according to the x and y coordinate planes. The "Goal" marker indicates the target position at (0,0). Consequently, the mobile platform stopped close to this benchmark in the initial tests. However, a noticeable divergence from the intended location is observed with successive iterations of the experiments.

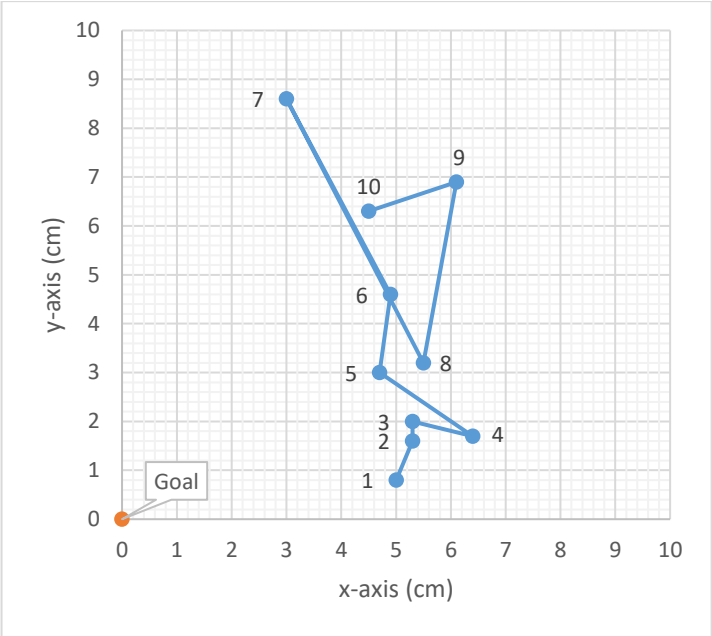


Figure 5.6. Endpoint Coordinates of the Accuracy Testing

The distances between the reached locations and the goal which represented in Figure 5.7 were computed based on the x and y coordinates, and arranged in accordance with the test order. Similarly, errors in distance are observed to increase with repeated tests, resulting in an average distance of 6.76 cm from the target location.

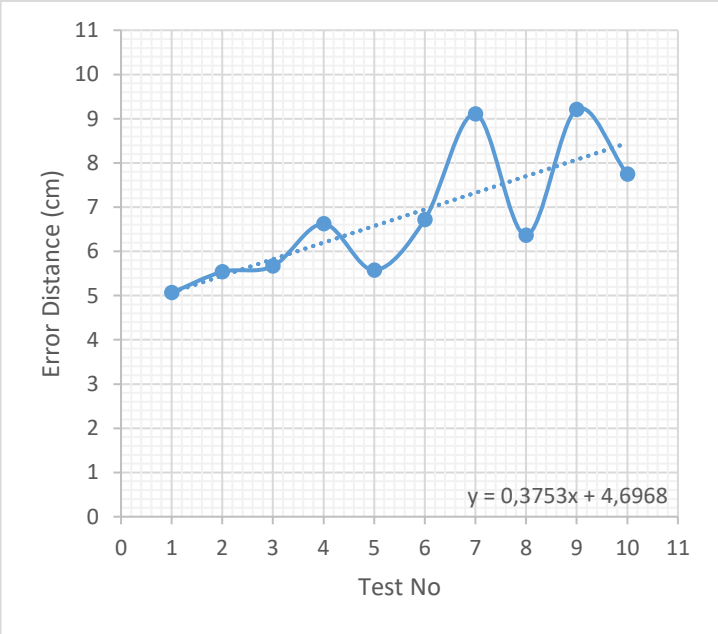


Figure 5.7. Distance Deviations of the Accuracy Testing

CHAPTER 6
CONCLUSIONS AND FUTURE RESEARCH

In this thesis, a cost-effective mobile platform equipped with an RGB-D camera and a 2D LiDAR have been presented. By integrating RTAB-Map and Hector SLAM, the robot achieves the capability of localization without relying on external odometry information. While LiDAR-based odometry is sufficient for executing basic commands toward the goal, further improvement can be made.

By combining the mapping capabilities of RTAB-Map with the navigation features of "move_base", the integrated system can effectively utilize the map for path planning, adapt to changes in the environment, and provide localization for successful robot navigation. By adding to TEB (Timed Elastic Band) local planner into the ROS Navigation Stack, the robot demonstrated reliable obstacle detection, effective path planning adjustments, and smooth trajectory modifications to prevent collision while preserving its navigation goal.

The greater variation in the y-axis direction for the positions reached during the test is attributed to the platform moving back and forth once to align itself with the x-axis of the orientation. Significantly, the distance error rises alongside the increasing number of tests and the average distance from the intended location is 6.76 cm.

The navigation system was assigned specific tasks with certain limitations:

- Although the Raspberry Pi provides an affordable and small-scale option, its processing power is relatively limited when compared to more advanced computing systems. This limitation can potentially impact the speed, efficiency, and real-time performance of SLAM algorithms, resulting in slower mapping and localization updates, decreased accuracy, or restricted capabilities.
- During the first operation of the system, a room in an unfamiliar environment was mapped by manual control. After that, when the navigation operation is launched, the loading of the 3D point cloud map takes time before starting the robot navigation.

- The platform could be lost while performing navigation due to the reliance only on LiDAR odometry. It is best to ensure that scan data is provided and restart the application from the initial position.
- As mobile robots heavily rely on battery power, the duration of the operation is limited.

Future research to improve the system would be mapping a remote computer which can potentially decrease the mapping and motion planning computation time. Since Raspberry Pi has experienced performance limitations, performing mapping and navigation tasks on a remote computer leads to faster and more reliable performance on the Raspberry Pi 4 platform. Furthermore, integrating the encoder and IMU sensors can improve the precision of odometry information. Also, the performance of different global planner algorithms combined with the TEB local planner can be investigated.

REFERENCES

Albers, F., Rösmann, C., Hoffmann, F., & Bertram, T. (2019). Online trajectory optimization and navigation in dynamic environments in ROS. In *Robot Operating System (ROS) The Complete Reference (Volume 3)* (pp. 241–274). Springer International Publishing. https://doi.org/10.1007/978-3-319-91590-6

Alsadik, B., & Karam, S. (2021). The Simultaneous Localization and Mapping (SLAM) - An Overview. *Surveying and Geospatial Engineering Journal*, *02*(01), 1–12. https://doi.org/10.38094/sgej1027

Bailey, T., & Durrant-Whyte, H. (2006). Simultaneous localization and mapping (SLAM): Part II. *IEEE Robotics and Automation Magazine*, *13*(3), 108–117. https://doi.org/10.1109/MRA.2006.1678144

Bresson, G., Alsayed, Z., Yu, L., & Glaser, S. (2017). Simultaneous Localization and Mapping: A Survey of Current Trends in Autonomous Driving. *IEEE Transactions on Intelligent Vehicles*, *2*(3), 194–220. https://doi.org/10.1109/TIV.2017.2749181

Chen, C., Zhu, H., Li, M., & You, S. (2018). A review of visual-inertial simultaneous localization and mapping from filtering-based and optimization-based perspectives. *Robotics*, *7*(3). https://doi.org/10.3390/robotics7030045

Chen, K. C., & Tsai, W. H. (2010). Vision-based autonomous vehicle guidance for indoor security patrolling by a sift-based vehicle-localization technique. *IEEE Transactions on Vehicular Technology*, *59*(7), 3261–3271. https://doi.org/10.1109/TVT.2010.2052079

ChristophRoesmann. (2020). *teb_local_planner*. ROS Wiki. http://wiki.ros.org/teb_local_planner

Cousins, S. (2010). ROS on the PR2. *IEEE Robotics and Automation Magazine*, *17*(3), 23–25. https://doi.org/10.1109/MRA.2010.938502

Das, S. (2018a). Robot localization in a mapped environment using Adaptive Monte Carlo algorithm. *International Journal of Scientific & Engineering Research*, *9*(10), 77–85.

Das, S. (2018b). *Simultaneous Localization and Mapping (SLAM) using RTAB-MAP*. 1–6. http://arxiv.org/abs/1809.02989

Donoso, F. A., Austin, K. J., & Mcaree, P. R. (2017). Three new Iterative Closest Point variant-methods that improve scan matching for surface mining terrain. *Robotics and Autonomous Systems*, *95*, 117–128. https://doi.org/10.1016/j.robot.2017.05.003

Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous localization and mapping: Part I. *IEEE Robotics and Automation Magazine*, *13*(2), 99–108. https://doi.org/10.1109/MRA.2006.1638022

Durrant-Whyte, H., Rye, D., & Nebot, E. (1996). Localization of Autonomous Guided Vehicles. In G. Giralt & G. Hirzinger (Eds.), *Robotics Research* (pp. 613–625). Springer London.

Endres, F., Hess, J., Sturm, J., Cremers, D., & Burgard, W. (2014). 3-D Mapping with an RGB-D camera. *IEEE Transactions on Robotics*, *30*(1), 177–187. https://doi.org/10.1109/TRO.2013.2279412

Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, *4*(1), 23–33. https://doi.org/10.1109/100.580977

Garcia-Fidalgo, E., & Ortiz, A. (2015). Vision-based topological mapping and localization methods: A survey. *Robotics and Autonomous Systems*, *64*, 1–20. https://doi.org/10.1016/j.robot.2014.11.009

Grisetti, G., Kümmerle, R., Stachniss, C., & Burgard, W. (2010). A Tutorial on Graph-Based SLAM. *IEEE Intelligent Transportation Systems Magazine*, *2*(4), 31–43. https://doi.org/10.1109/MITS.2010.939925

Grisetti, G., Stachniss, C., & Burgard, W. (2007). *Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters*. *23*(1), 34–46.

Guimarães, R. L., de Oliveira, A. S., Fabro, J. A., Becker, T., & Brenner, V. A. (2016). ROS navigation: Concepts and tutorial. *Studies in Computational Intelligence*, *625*(May 2018), 121–160. https://doi.org/10.1007/978-3-319-26054-9_6

Harik, E. H. C., & Korsaeth, A. (2018). Combining hector SLAM and artificial potential field for autonomous navigation inside a greenhouse. *Robotics*, *7*(2). https://doi.org/10.3390/robotics7020022

Houssein, E., Harik, C., & Korsaeth, A. (2019). *The Heading Weight Function : A Novel LiDAR-Based Local Planner for Nonholonomic Mobile Robots The Heading Weight Function : A Novel LiDAR- Based Local Planner for Nonholonomic Mobile Robots*. *August*. https://doi.org/10.3390/s19163606

Joseph, L. (2018). Robot Operating System (ROS) for Absolute Beginners. In *Robot Operating System (ROS) for Absolute Beginners*. Springer. https://doi.org/10.1007/978-1-4842-7750-8

Julier, S. J., & Uhlmann, J. K. (2004). Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, *92*(3), 401–422. https://doi.org/10.1109/JPROC.2003.823141

JunyaHayashi. (2021). *Setting up your robot using tf*. ROS Wiki.

Kohlbrecher, S., Von Stryk, O., Meyer, J., & Klingauf, U. (2011). A Flexible and Scalable SLAM System with Full 3D Motion Estimation. *9th IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR 2011*, 155–160. https://doi.org/10.1109/SSRR.2011.6106777

Labbé, M., & Michaud, F. (2018a). Long-term online multi-session graph-based SPLAM with memory management. *Autonomous Robots*, *42*(6), 1133–1150. https://doi.org/10.1007/s10514-017-9682-5

Labbé, M., & Michaud, F. (2018b). RTAB-Map as an Open-Source Lidar and Visual SLAM Library for Large-Scale and Long-Term Online Operation. *Journal of Field Robotics*, *36*(2), 416–446. http://introlab.github.io/rtabmap/

Lu, F., & Milios, E. (1997). Globally Consistent Range Scan Alignment for

Environment Mapping. *Autonomous Robots*, *4*(4), 333–349. https://doi.org/10.1023/A:1008854305733

Marin-plaza, P., Hussein, A., Martin, D., & Escalera, A. De. (2018). *Global and Local Path Planning Study in a ROS-Based Research Platform for Autonomous Vehicles*. *2018*.

MathieuLabbe. (2023a). *Mapping and Navigation with Turtlebot*. ROS Wiki. http://wiki.ros.org/rtabmap_ros/Tutorials/MappingAndNavigationOnTurtlebot

MathieuLabbe. (2023b). *Setup RTAB-Map on Your Robot!* ROS Wiki. http://wiki.ros.org/rtabmap_ros/Tutorials/SetupOnYourRobot

Miles, R. (2012). Start Here! Learn the Kinect API. In *Pearson Education*.

Morgan Quigley, Brian Gerkey, W. D. S. (2015). *Programming Robots with ROS: a practical introduction to the Robot Operating System*. O'Reilly Media, Inc.

Naotunna, I., & Wongratanaphisan, T. (2020). Comparison of ROS Local Planners with Differential Drive Heavy Robotic System. *International Conference on Advanced Mechatronic Systems, ICAMechS*, *2020-Decem*, 1–6. https://doi.org/10.1109/ICAMechS49982.2020.9310123

NickLamprianidis. (2018). *costmap_2d*. ROS Wiki. http://wiki.ros.org/costmap_2d

Nurmaini, S., & Pangidoan, S. (2018). Localization of Leader-Follower Robot Using Extended Kalman Filter. *Computer Engineering and Applications Journal*, *7*(2), 91–104. https://doi.org/10.18495/comengapp.v7i2.253

Oliver, A., Kang, S., Wünsche, B. C., & MacDonald, B. (2012). Using the Kinect as a navigation sensor for mobile robotics. *ACM International Conference Proceeding Series*, *August 2015*, 509–514. https://doi.org/10.1145/2425836.2425932

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., & Ng, A. (2009). ROS: an open-source Robot Operating System. *ICRA Workshop on Open Source Software*, *3*(3.2), 5. https://doi.org/10.1109/IECON.2015.7392843

Quinlan, S., & Oussama, K. (1993). Elastic Bands: Connecting Path Planning and Control. In *IEEE International Conference on Robotics and Automation* (Vol. 2, pp. 802–807).

Rösmann, C., Feiten, W., Wösch, T., Hoffmann, F., & Bertram, T. (2012). Trajectory modification considering dynamic constraints of autonomous robots. *7th German Conference on Robotics, ROBOTIK 2012*, 74–79.

Shanghai Slamtec Co., L. (2009). *RPLIDAR A1 Introduction and Dataset*. 1–16. https://bucket-download.slamtec.com/d1e428e7efbdcd65a8ea111061794fb8d4ccd3a0/LD108_SLAMTEC_rplidar_datasheet_A1M8_v3.0_en.pdf

Smith, R., & Cheeseman, P. (1987). On the Representation and Estimation of Spatial Uncertainty. *The International Journal of Robotics Research*, *5*. https://doi.org/10.1177/027836498600500404

Smith, R., Self, M., & Cheeseman, P. (1990). Estimating Uncertain Spatial Relationships in Robotics. In I. J. Cox & G. T. Wilfong (Eds.), *Autonomous*

Robot Vehicles (pp. 167–193). Springer New York. https://doi.org/10.1007/978-1-4613-8997-2_14

StefanKohlbrecher. (2011). *How to set up hector_slam for your robot*. ROS Wiki. http://wiki.ros.org/hector_slam/Tutorials/SettingUpForYourRobot

Talwar, D., & Jung, S. (2019). Particle Filter-based Localization of a Mobile Robot by Using a Single Lidar Sensor under SLAM in ROS Environment. *International Conference on Control, Automation and Systems*, *2019-Octob*(Iccas), 1112–1115. https://doi.org/10.23919/ICCAS47443.2019.8971555

Ullah, I., Shen, Y., Su, X., Esposito, C., & Choi, C. (2020). A Localization Based on Unscented Kalman Filter and Particle Filter Localization Algorithms. *IEEE Access*, *8*, 2233–2246. https://doi.org/10.1109/ACCESS.2019.2961740

Zacwitte. (2016). *costmap_converter*. ROS Wiki. http://wiki.ros.org/costmap_converter

Zhang, F., Li, S., Yuan, S., Sun, E., & Zhao, L. (2017). Algorithms analysis of mobile robot SLAM based on Kalman and particle filter. *2017 9th International Conference on Modelling, Identification and Control (ICMIC)*, 1050–1055. https://doi.org/10.1109/ICMIC.2017.8321612

Zhang, X., Lai, J., Xu, D., Li, H., & Fu, M. (2020). 2D Lidar-Based SLAM and Path Planning for Indoor Rescue Using Mobile Robots. *Journal of Advanced Transportation*, *2020*. https://doi.org/10.1155/2020/8867937

Zhang, X., Lu, G., Fu, G., Xu, D., & Liang, S. (2023). SLAM Algorithm Analysis of Mobile Robot Based on Lidar. *2019 Chinese Control Conference (CCC)*, 4739–4745.

Zheng, K. (2021). ROS Navigation Tuning Guide. *Studies in Computational Intelligence*, *962*, 197–226. https://doi.org/10.1007/978-3-030-75472-3_6