INDOOR NAVIGATION ALGORITHM OPTIMIZATION ON ROS

A PROJECT REPORT

Submitted in partial fulfillment of the requirement for the award of the Degree of

BACHELOR OF TECHNOLOGY IN ELECTRONICS AND COMMUNICATION ENGINEERING

by

P. S. N. SURYAVAMSI (15BEC1127)

Under the Guidance of

Dr. V BERLIN HENCY

SCHOOL OF ELECTRONICS ENGINEERING
VELLORE INSTITUTE OF TECHNOLOGY
CHENNAI - 600127

May 2019

CERTIFICATE

This is to certify that the Project work titled "***INDOOR NAVIGATION ALGORITHM OPTIMIZATION ON ROS***" that is being submitted by ***P.S.N. SURYAVAMSI (15BEC1127)*** is in partial fulfillment of the requirements for the award of **Bachelor of Technology in Electronics and Communication Engineering**, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.

V Berlin Hency
Guide

The Project Report is satisfactory / unsatisfactory

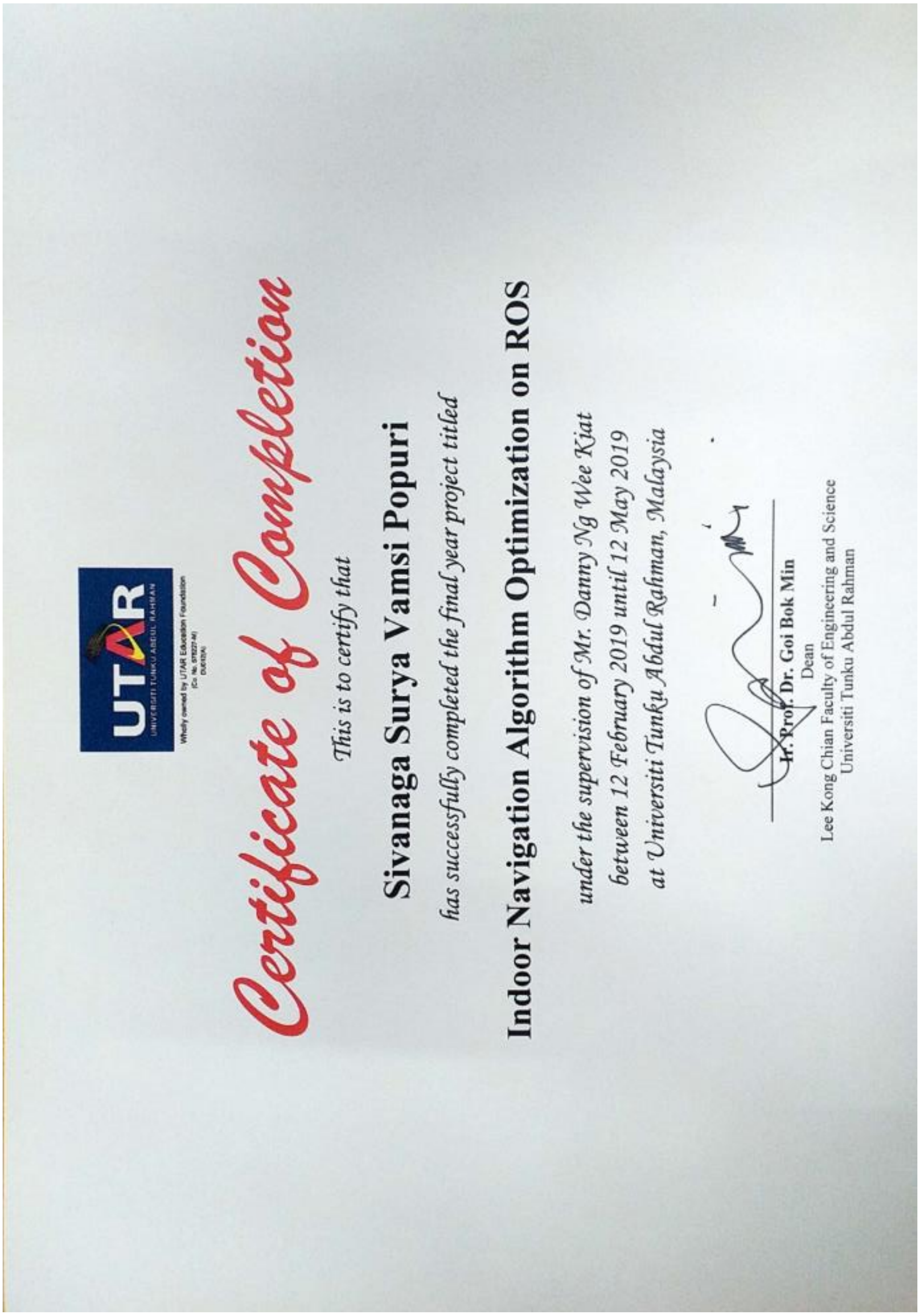
Internal Examiner **External Examiner**

Approved by

PROGRAM CHAIR
B. Tech. (ECE)

DEAN
School of Electronics Engineering

BONAFIDE CERTIFICATE of UTAR, Sg Long, Malaysia



UTAR
UNIVERSITI TUNKU ABDUL RAHMAN

Wholly owned by UTAR Education Foundation
(Co. No. 578227-M)
DU012(A)

Certificate of Completion

This is to certify that

Sivanaga Surya Vamsi Popuri

has successfully completed the final year project titled

Indoor Navigation Algorithm Optimization on ROS

under the supervision of Mr. Danny Ng Wee Kiat
between 12 February 2019 until 12 May 2019
at Universiti Tunku Abdul Rahman, Malaysia

[illegible]

Ir. Prof. Dr. Goi Bok Min
Dean
Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

ACKNOWLEDGEMENTS

I would like to thank **VIT University** for giving me the prestigious opportunity to carry out the project as a Semester Abroad Program at *UTAR (Universiti Tunku Abdul Rahman),* Malaysia, a partner university of VIT.

I am extremely grateful to **Dr Sivasubramanian**, Dean of the School of Electronics Engineering, and **Dr Vetrivelan P**, Program Chair, School of Electronics Engineering, VIT Chennai for their support for the entire course of the course of the project.

I would also like to express my sincere most gratitude to my project Guide, **Dr. V Berlin Hency**, School of Electronics Engineering, VIT Chennai for her consistent motivation and valuable mentoring offered to me for the entire course of the project work.

P.S.N.SURYAVAMSI
(15BEC1127)

ABSTRACT

Robots are used in both the industrial and non-industrial environments. The advancement of mobile robotics allows for multiple use case of mobile robots such as warehouse management and indoor service. All these robots can work autonomously with minimum human intervention. For these robots to work autonomously, environment sensors such as laser scanning rangefinders and depth cameras, simultaneous localization and mapping (SLAM) algorithm, path planners are implemented together in the robot. Multiple algorithms need to communicate with each other to ensure smooth operation of an autonomous mobile robot. Robot Operating System (ROS) is one of the widely used platforms to facilitate communication between processes in research.

The aim of this project to apply a navigation algorithm called the TEB (Timed Elastic Band) algorithm on a service robot in an indoor space, by means of the ROS (Robot Operating System) platform, by means of the open source package library called teb_local planner. It is originally designed to work on the robot built by its developers of the package. The task to be done is the customization of the algorithm for proper functioning on the robot being used and understand the effect of the tuning parameters responsible for the working of the algorithm.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ROS	Robot Operating System
SLAM	Simultaneous Localization And Mapping
TEB	Timed Elastic Band
RVIZ	ROS Visualization
AMCL	Adaptive Monte Carlo Localization
DWA	Dynamic Window Approach
PIC	Peripheral Interface Controller
DT	Discrete Time
TF	Transform
PC	Personal Computer
HC	Homotopy Class
OS	Operating System
SSH	Secure Shell
LAN	Local Area Network
MCU	Microcontroller
LASER	Light Amplification by Stimulated Emission of Radiation
DC	Direct Current
GTG	Go to Goal
WIFI	Wireless Fidelity
EB	Elastic Band
GUI	Graphical User Interface
LI-ION	Lithium Ion
CMD	Command
DFS	Depth First Search
2-D	Two Dimensional
DD	Differential Drive
USB	Universal Serial Bus
OA	Obstacle Avoidance
GT	Goal Tolerance
CM	Cost Map
LS	Laser Scan
NH	Non-Holonomic

CHAPTER 1
INTRODUCTION

The concept of indoor robot navigation is crucial when it comes to service robotics and autonomous systems. Robust motion planning strategies are needed for robots to operate in highly dynamic environments. As the kind of robot being used here is an indoor service robot, inaccuracies in the environment are inevitable. For this purpose, online trajectory planning is the best approach as it immediately responds to perturbations in the environment during runtime. Many algorithms have been proposed and implemented to solve the problem of robot navigation. In this project, however, the TEB (Timed Elastic Band) algorithm is being used as it is computationally less expensive, allows the user to customize its behavior and also generates accurate trajectories for the robot to take.

1.1 Objectives

The following are the objectives of this project:

- To study and understand the numerical tuning parameters behind the working of the TEB navigation algorithm.
- To realize the working of the TEB algorithm by using the teb_local_planner package library first on a robot in a virtual environment.
- To identify variable parameters which form the basis of the working of the algorithm and understand their effects on the behavior of the algorithm and also the mobile robot.
- To customize behavior of the algorithm on the virtual simulator and the dimensions of the robot, by tuning the variable parameters of the package library.
- To implement the algorithm on an actual indoor service robot in a selected indoor environment.

1.2 Background and Literature Survey

In order to incorporate autonomous navigation of a mobile robot, numerous algorithms have been proposed. Chi-Chung Chou et al. have proposed a navigation scheme based on analyzing and verifying the next possible position for the robot to take before proceeding further [5], while L. Jaillet et al. suggest a motion planning technique based on sampling and roadmaps [6].

Francisco A.X. Da Mota et al. propose a navigation system in which a network of RFID plates are placed like a grid in a room, and the robot has an RFID reader using which it can estimate its position based on the RFID plate it encounters [4]. On the other hand, J. Mattingley et al. propose a time-based solving of optimization constraints [7].

In [8], Ross A Knepper et al. propose a scheme in which the approach avoids expensive collision tests on a given path by sensing that the entire geometry of this path has effectively already been tested on a combination of other paths, while Woojin Chung et al. propose a navigation scheme considering regions that are obstructed by obstacles [9]. Mohamed M Atia et. Al [10] propose a self-navigation system using various sensors and without the use of GPS.

Many techniques are proposed for localization as well. In [11], a map based localization is proposed, while in [12 and [24], neuro-fuzzy and sensor-based localization is done, and in [15], multiple ultrasonic sensors are used for localization.

In the Elastic Band algorithm presented in [1], the robot can navigate from an initial pose to an assigned goal by taking a short and smooth path generated by a global planner. In a dynamic environment where obstacles appear at random points of time, the path can deform itself like an elastic band such that the navigating robot doesn't collide with obstacles, and the overall nature of the initial path that is generated by the global planner is preserved.

[3] illustrates a modified version of the aforementioned Elastic Band algorithm, called the Timed-Elastic-Band (TEB) algorithm. The new method overcomes the disadvantage of the old one by considering the motion-specific constraints of the robot such as velocity and acceleration limits, which weren't considered in [1]. Also, in the new method, the fastest path to reach the goal is considered, rather than the shortest path by distance i.e. temporal

information of the robot’s trajectory is considered. Thus, the algorithm generates a trajectory which ensures the least time taken to reach the goal, avoidance of obstacles, and the adherence to the kino-dynamic constraints of the robot, the primary conditions followed by the algorithm.

However, the common problem with [1] and [3],is that although the path from start to goal is elastic like a band which stretches due to obstacles, it cannot transit across obstacles; it keeps on stretching. Thus, [2] proposes an even modified version of the TEB algorithm in which consider alternative admissible trajectories. If the current optimum trajectory is affected by an obstacle and stretches, the algorithm shifts the optimum solution to a different trajectory which still obeys the conditions mentioned above, thus becoming the new optimum. [2] includes the concept of homotopy classes, in which feasible alternative trajectories are considered as part of a common class such that the trajectories can be interchanged based on obstacles to become updated optimum solutions immediately.

An open source package library called **teb_local_planner** was developed which can be used on ROS (Robot Operating System) to realise the TEB algorithm both on a virtual environment and in real-time as well [41], and is based on the algorithm presented in [2]. The algorithm is iterative, and makes use of certain tuning parameters which impact the generation of trajectories in different conditions such as obstacle proximity, the organization of the indoor environment, the velocity commands given to the robot, etc. The tuning parameters are obtained from the underlying control equations which govern the motion of the robot and also the nature of the trajectory formed in the environment. In tweaking these parameters, the algorithm behaves in a different manner, and the effect of tuning parameters on the behavior of the algorithm can be understood. With this knowledge, the user can customize the parameters’ values in order to alter the performance of the algorithm as desired, which is the main focus of this project.

Table 1.1 shows the literature survey of the research which contains the information drawn from different sources used for reference.

Table 1.1 Literature Survey

S. No	**Name of article**	**Name of journal**	**Information included**
1.	Real time modification based on collision free paths [1]	Stanford University, Stanford, CA, USA, 1995	Planning based on separation from obstacles
2.	Integrated Online Trajectory Planning and Optimization in Distinctive Topologies [2]	International Journal of Electrical, Computer, Energetic, Electronic and Communication Engineering Vol:7, No:12, 2013	Planning of multiple trajectories using Homotopy Class
3.	Trajectory Modification Considering Dynamic Constraints of Autonomous Robots [3]	7th German Conference on Robotics, Germany, Munich, 2012, pp 74–79	Minimizing time, obstacle and constraints based objective cost function.
4.	Localization and Navigation for Autonomous Mobile Robots Using Petri Nets in Indoor Environments [4]	IEEE Access, 2018, Volume 6	RFID-network based navigation
5.	Characterizing Indoor Environment for Robot Navigation Using Velocity Space Approach With Region Analysis and Look-Ahead Verification [5]	IEEE Transactions on Instrumentation and Measurement (Volume: 60 , Issue: 2 , Feb. 2011)	A-star search based look-ahead verification and velocity control
6.	Path deformation roadmaps: Compact graphs with useful cycles for motion planning [6]	The International Journal of Robotics Research, vol. 27, no. 11-12, pp. 1175–1188, 2008	Sampling-based motion planning
7.	Receding Horizon Control: Automatic Generation of High-Speed Solvers [7]	IEEE Control Systems Magazine, Vol. 31, No. 3, pp. 52-65, 2011	Constrained optimization solving using time-horizon
8.	Toward a deeper understanding of motion alternatives via an equivalence relation on local paths [8]	Int. J. Robot. Res. 31 (2) (2012) 168–187	Previous path collision tests on new paths.
9.	Safe Navigation of a Mobile Robot Considering Visibility of Environment [9]	IEEE Transactions on Industrial Electronics (Volume: 56 , Issue: 10 , Oct. 2009)	Collision avoidance based on occluded regions
10.	Integrated Indoor Navigation System for Ground Vehicles With Automatic 3-D Alignment and Position Initialization [10]	IEEE Transactions on Vehicular Technology (Volume: 64 , Issue: 4 , April 2015)	Self-alignment without navigation system

S. No	Name of Article	Name of Journal	Information Included
11.	Topological Indoor Localization and Navigation for Autonomous Mobile Robot [11]	IEEE Transactions on Automation Science and Engineering (Volume: 12 , Issue: 2 , April 2015)	Map based localization
12.	Indoor Intelligent Mobile Robot Localization Using Fuzzy Compensation and Kalman Filter to Fuse the Data of Gyroscope and Magnetometer [12]	IEEE Transactions on Industrial Electronics (Volume: 62 , Issue: 10 , Oct. 2015)	Localization based on sensor data and thresholding.
13.	Obstacle Detection and Avoidance by a Mobile Robot Using Probabilistic Models [13]	IEEE Latin America Transactions (Volume: 13 , Issue: 1 , Jan. 2015)	Gaussian function-based obstacle estimation
14.	Receding Horizon Control for Convergent Navigation of a Differential Drive Mobile Robot [14]	IEEE Transactions on Control Systems Technology (Volume: 25 , Issue: 2 , March 2017)	Cost function without local minima
15.	Dynamic Ultrasonic Hybrid Localization System for Indoor Mobile Robots [15]	IEEE Transactions on Industrial Electronics (Volume: 60 , Issue: 10 , Oct. 2013)	Localization using multiple ultrasonic sensors
16.	Topological and geometric techniques in graph-search based robot planning [16]	Ph.D. dissertation, University of Pennsylvania, January 2012	Motion planning using graph-search
17.	Discrete-Status-Based Localization for Indoor Service Robots [17]	IEEE Transactions on Industrial Electronics (Volume: 53 , Issue: 5 , Oct. 2006)	Discrete Petri-net based localization
18.	Autonomous Mobile Robot Navigation Using Passive RFID in Indoor Environment [18]	IEEE Transactions on Industrial Electronics (Volume: 56 , Issue: 7 , July 2009)	Navigation based on Radio frequency Grids
19.	Mobile Robot Navigation Through a Hardware-Efficient Implementation for Control-Law-Based Construction of Generalized Voronoi Diagram	IEEE/ASME Transactions on Mechatronics (Volume: 16 , Issue: 6 , Dec. 2011)	Modified Voronoi Diagram
20.	Accurate 3-D Position and Orientation Method for Indoor Mobile Robot Navigation Based on Photoelectric Scanning [20]	IEEE Transactions on Instrumentation and Measurement (Volume: 64 , Issue: 9 , Sept. 2015)	Rotary laser based position estimation
21.	Ultra-wideband indoor navigation system [21]	IET Radar, Sonar & Navigation (Volume: 6 , Issue: 5 , June 2012)	Radio frequency based navigation in ultra wideband

S No.	Name of Article	Name of Journal	Information Included
22.	Dynamic Wireless Indoor Localization Incorporating With an Autonomous Mobile Robot Based on an Adaptive Signal Model Fingerprinting Approach [22]	IEEE Transactions on Industrial Electronics (Volume: 66 , Issue: 3 , March 2019)	ASMF fingerprint based localization
23.	Signage System for the Navigation of Autonomous Robots in Indoor Environments [23]	IEEE Transactions on Industrial Informatics (Volume: 10 , Issue: 1 , Feb. 2014)	Signage based RFID system for navigation
24.	Behavior-based neuro-fuzzy controller for mobile robot navigation [24]	IEEE Transactions on Instrumentation and Measurement (Volume: 52 , Issue: 4 , Aug. 2003)	Threshold based control commands
25.	Sensor-Deployment Strategies for Indoor Robot Navigation [25]	IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans (Volume: 40 , Issue: 2 , March 2010)	Navigation Configuration error correction

1.3 Need for a navigation algorithm

It’s necessary that an autonomous device navigate on its own in an environment. If a robot has a specific task which involves going to a particular place, it must navigate itself to that place, while being aware of and avoiding dangerous situations such as collisions and unsafe conditions (temperature, radiation, exposure to weather, etc.)

A reliable and robust navigation algorithm implies that the robot can accurately move from a desired start pose to a desired goal pose. This can be used as a platform to build more applications not limited to indoor environments, such as delivery robots, autonomous self-driving cars, etc. In other words, this platform can open doors for various reliable and real-time applicable robotic applications in many different fields.

1.4 Contribution of the Project

The primary focus of this research is on the application of the TEB navigation algorithm on a mobile indoor service robot through ROS (Robot Operating System), using the teb_local_planner package library. The objective is to optimize the algorithm's performance on the robot by customizing the parameters involved in the teb_local_planner package library by tuning their values and understanding their effects on the performance of the algorithm and thus the behavior of the indoor robot.

1.5 Organization of the Report

The rest of the report is organized as follows:

- Chapter 2 contains the theory and background behind concepts used in the project.
- Chapter 3 lists the methodology and procedures followed in implementing the project.
- Chapter 4 gives the technical specifications involved both in terms of hardware and software used.
- Chapter 5 describes the implementation of the navigation algorithm on ROS.
- Chapter 6 presents the results obtained after carrying out the implementation.
- Chapter 7 gives the conclusion and suggests possible work to be done in the future.

CHAPTER 2
THEORY AND BACKGROUND

This Chapter describes the theory and background behind the concepts used in this project.

2.1 ROS (Robot Operating System)

The Robot Operating System (ROS) is a flexible platform for designing robotic applications. It consists of libraries, standards and tools intended to simplify the task of inventing complicated and robust robotic behavior across various platforms. The reason it is needed is because it’s hard to create a general-purpose software for robots that’s completely robust. According to a robot, issues that are normal to humans often are different for different tasks and places. Thus, it’s hard a single individual, institute or laboratory to implement applications on their own [40].

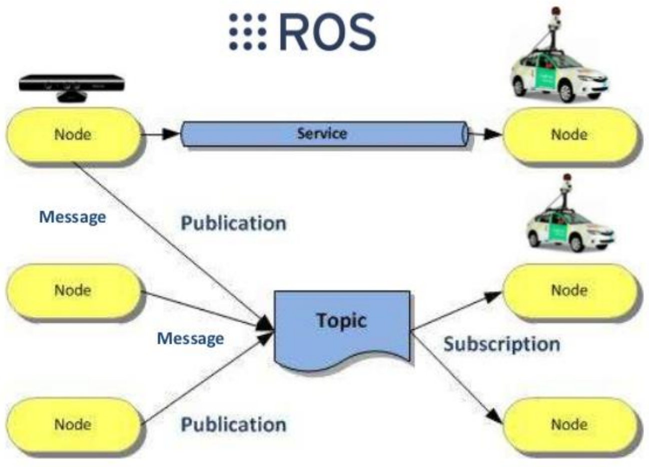


Fig 2.1: Framework of ROS

As a result of this, ROS was built from a basic level to foster joint robotic applications, as shown in Fig 2.1. For instance, one department might have experts in obtaining maps of indoor regions, and could be a potential supplier of a world-class map producing system. Another department might have experts at utilizing those maps for navigation, and another department might have come up with a method that uses ultrasonic technology which works good for detecting obstacles in a region. ROS was designed in order for groups and departments like these to work with each other, exchange ideas and build upon each other's work.

2.2 Features of ROS

ROS has user-friendly features which support the development of various applications. These features are described below [40].

1. **Peer-to-peer communication**: Robotic systems with multiple connections may have numerous on-board computers (for parallel computing) connected via a network. Too much overcrowding would occur in one connection if run on a central master. This issue can be overcome by using peer-to-peer communication. In ROS, the inter-peer system coupled with a buffer system and a chief system (in ROS, it’s called a ‘master’) allows each individual node to communicate with each other in the desired manner.

2. **Free and open-source**: Being open-source platform provides reuse of already existing functions provided by the many other ROS users. Their code is supplied in repositories as "stacks". Other people have developed amazing capabilities for robots that have been "open-sourced" and are relatively easy to add incrementally using the ROS development environment.

3. **Simple**: Developers of ROS include algorithms and drivers in executable files. This means that reusability is more, and also, the size is lesser. This approach makes ROS an easy platform to use, and the complicated material is only the package libraries. With this, unit wise testing is also possible and the developed systems can be completely independent of any another system.

4. **Language neutrality**: Because of its neutrality, ROS can be programmed in various programming languages like C++, Python, Java, etc. The specification of ROS comes into play at the messaging layer. Inter-peer messages are made in XML-RPC, which is present in various languages. For a new language to be supported, either re-wrapping of C++ classes is done (e.g. GNU Octave software) or new classes are written enabling messages to be made. IDL (Interface Definition Language) is used to communicate the messages.

2.3 The TEB algorithm

2.3.1 The TEB objective function

The TEB (Timed Elastic Band) algorithm's main objective is to generate a trajectory to be taken by the robot by following these conditions: the robot must reach the goal in minimal time by avoiding obstacles and adhering to the motion restrictions of the robot such as acceleration limits and speed limits. The TEB problem is formulated as an objective function as shown below [2, 3].

$$f(B) = \sum_k \gamma_k f_k(B) \quad \textbf{(1)}$$

The function f(B) is an objective cost function which needs to be minimized i.e. a minimum cost value needs to be obtained. Here, B is a tuple containing two elements: a set of 'n' consecutive poses from start to goal that is to be taken by the robot, and a set of 'n-1' time intervals between two consecutive poses which the robot takes. The gamma term represents the weights. Individual objective functions $f_k(B)$ represent the sub-objective functions based on each category to be considered such as obstacle separation, robot dimensions, etc. The objective is to find the optimum tuple B^* which is the tuple solution which gives us the minimum value of f(B) and is the path which is ultimately taken by the robot to reach its assigned goal. The parameters and weights which constitute this cost function are discussed in later sections.

2.3.2 Homotopy Class

As presented in [2], the optimum trajectory needs to transit across obstacles in order for the robot to reach the goal in minimal time. If the current trajectory is not feasible due to increased time, the optimizer should immediately shift to another trajectory which also follows the conditions of the TEB discussed in section 3.1.1. Whether the new trajectory is admissible or not is decided by the concept of Homotopy class. According to [2], two trajectories τ1 and τ2 having common start and goal points zs and zg, are homotopic w.r.t each other only if one can be transformed into the other without passing through any obstacles. A homotopy class is the set of all trajectories which follow the aforementioned condition with every other trajectory. Here, τ denotes the robot's path in terms of a sequence of robot positions in a 2D plane which connect the robot's current position to the goal. The principle involved in finding a homotopy class is presented as follows.

Equation (2), known as the H-signature formula, provides a unique complex number to trajectories of the same homotopy class. The following equation which maps a given trajectory τ to the value of H is described below, known as the H-signature [3].

$$H(\tau) = \sum_{k=0}^{N-1} (H_s(z_k, z_{k+1})) \quad \textbf{(2)}$$

Where z_k is one of the points on the complex plane which is a part of the path from start to goal such that $z_k = x_k + iy_k$. If k =1, it’s the start pose and if k = N, it’s the goal pose. Here, the function in the summation term is given as follows [2].

$$H_s(z_k, z_{k+1}) = \sum_{l=1}^{R} A_l (\ln(z_{k+1} - \varepsilon_l) - \ln(z_k - \varepsilon_l)) \quad \textbf{(3)}$$

$$\text{Where } A_l = \frac{f_o(\varepsilon_l)}{\prod_{j=1, j\neq l}^{R} (\varepsilon_l - \varepsilon_j)} \quad \textbf{(4)}$$

Where $\varepsilon_l \in O_l$, $\forall l = 1, 2, . . . , R$ represent the positions of R obstacles in the entire 2D environment of the robot i.e. the plane O in the 2D space. The suggested H-signature tells us whether different trajectories come under the same homotopy class which is satisfied if every H-signature is identical.

2.3.3 Trajectory Planning Algorithm

With the robot’s current position zs, goal zg and the set of obstacle regions O = {O_l *such that* l = 1, 2, 3,. . . , R}, an exploration graph G = {V, E} is created in order to gather an initial subset of feasible routes. V represents the list of possible robot poses, and E represents the obstacles in the 2D space. Based on G, its easy paths between zs and zg are obtained by a depth first search supported by a visited list. Every path’s H-signature is obtained from **(2)** and is added to the set of members of known signatures in case it is not a member yet. Path candidates with similar H-signature are deleted. In the algorithm used by the optimizer [3], the set B contains those points already visited such that after reaching the goal, B consists of the complete trajectory from zs to zg. This path candidate is matched with homologue duplicates in H in steps 10 and 11, solved using **(2)**. If its homology class is new, the associated trajectory for the optimization problem is now started and initiated from the path B, the next time the optimizer begins again, for a warm start up.

Input: G - the acyclic graph, B - a list of visited points containing only zs, zg- goal vertex, T- the set of the trajectory; H - the set containing H-signatures

Output: Updated set of feasible trajectories and their H-signatures

```
Step 1:  Function DEPTHFIRSTSEARCH(B, G, zg, H, T)
Step 2:      if maximum size of T reached,
Step 3:          return
Step 4:      b ← B.back().                                   Lastly visited vertex
Step 5:      for each adjacent vertex v of vertex b in G do
Step 6:          if v in B then.                             Already seen
Step 7:              continue
Step 8:          if v is zg then.                            GOAL!
Step 9:              Include v in B.                         Finalize trajectory
Step 10:             h ←CALCHSIG(B).                         Equation (2)
Step 11:             if h not in H then
Step 12:                 x ←INITTRAJECTORY(B).
Step 13:                 T.append(x). Save complete trajectory
Step 14:                 H.append(h). Store H-signature
Step 15:                 break
Step 16:     for each adjacent vertex v of vertex b in G do
Step 17:         if v in B or v is zg then.                  Already seen or goal reached
Step 18:             continue
Step 19:         Include v in B.
Step 20:         DEPTHFIRSTSEARCH( B,G, zg, H, T).           Recursive.
Step 21:         Remove v from B.
Step 22: return
```

2.4 Summary

In this chapter, the theory and concepts behind the project were discussed. The definition of ROS (Robot Operating System) and its user-friendly features which support the feasibility of multiple applications were explained. The underlying principles and control equations behind the working of the TEB navigation algorithm were discussed and the steps followed by the algorithm were illustrated.

CHAPTER 3
METHODOLOGY AND PROCEDURE

This Chapter describes the methodology and procedure followed for the application and optimization of the indoor robot navigation algorithm.

3.1 Methodology Used

In order to implement the TEB (Timed Elastic Band) algorithm, the teb-local-planner package library was developed and is available online for open source download [41]. This package executes the TEB algorithm which is based on the collaborated functioning of many tunable numerical parameters.

For instance, there are parameters related to the robot's velocity and acceleration limits (in m/s and m/s^2 respectively), parameters pertaining to the distance of obstacles from the robot's current position (in meters), and so on. These parameters can be tweaked, and this will result in the algorithm behaving in a different manner every time the tweaking is done. While deploying the TEB algorithm, the user need not worry about the backend source codes involved in running the algorithm. The user can fine-tune the parameters that impact the algorithm during run-time in order to understand observable changes in behavior of the algorithm and hence, the behavior of the robot.

Initially, the implementation of the TEB algorithm and understanding the effects of parameter tweaking are done on a simulated environment. Using a virtual robot, a map of this environment using LASER scan data is obtained. The teb-local-planner package library is included to the navigation stack used by the robot to move. A start and goal pose in the environment are assigned, and how the algorithm initially functions is observed. Since the algorithm may not necessarily work as desired for the robot being used, it needs to be modified by tweaking the parameters, or in other words, optimized until the performance is as desired. With the new set of parameters, the algorithm is later tested on a real robot in an actual indoor environment, thus verifying the optimization.

By default, the parameters of the package are set such that the algorithm works well only for the robot that was used by the developers of the TEB package. In order to make the algorithm work for the virtual robot, the parameters are tweaked until the virtual robot follows the algorithm to the best extent. The parameters that cause a major impact on the behavior of the robot and trajectory generation are recorded and noted. Further tweaking and adjustments of said parameters are done in order to make the algorithm work for a robot in real-time, in an actual indoor environment.

3.2 Design Approach

The design approach of the project is described as follows.

- The simulation of the robot is carried out on two software tools, GAZEBO and RVIZ, two open-source tools.
- The former is used to visualize a three-dimensional virtual environment in which the mobile robot can navigate, and the latter is used for generating a map of said environment and also assigning start and goal poses for the robot to take, and to also visualize the trajectory that is taken by the robot from start to goal.
- The robot model used in the simulation is the TURTLEBOT3 – burger [37], a mobile robot equipped with a laser scanner to interact with its environment to detect obstacles and also generating a map.
- The robot also has an on-board microcontroller to give velocity commands based on its position, to the robot for motion.
- This robot is available as an open source package that can be downloaded from [37].

3.3 Block Diagram

Figure 3.1 presents a block diagram describing the tools and components used for implementing the navigation algorithm on ROS. In the block diagram, “Move-Base” is a launch file which consists of different individual components or “nodes” [41] responsible for carrying out the navigation process.

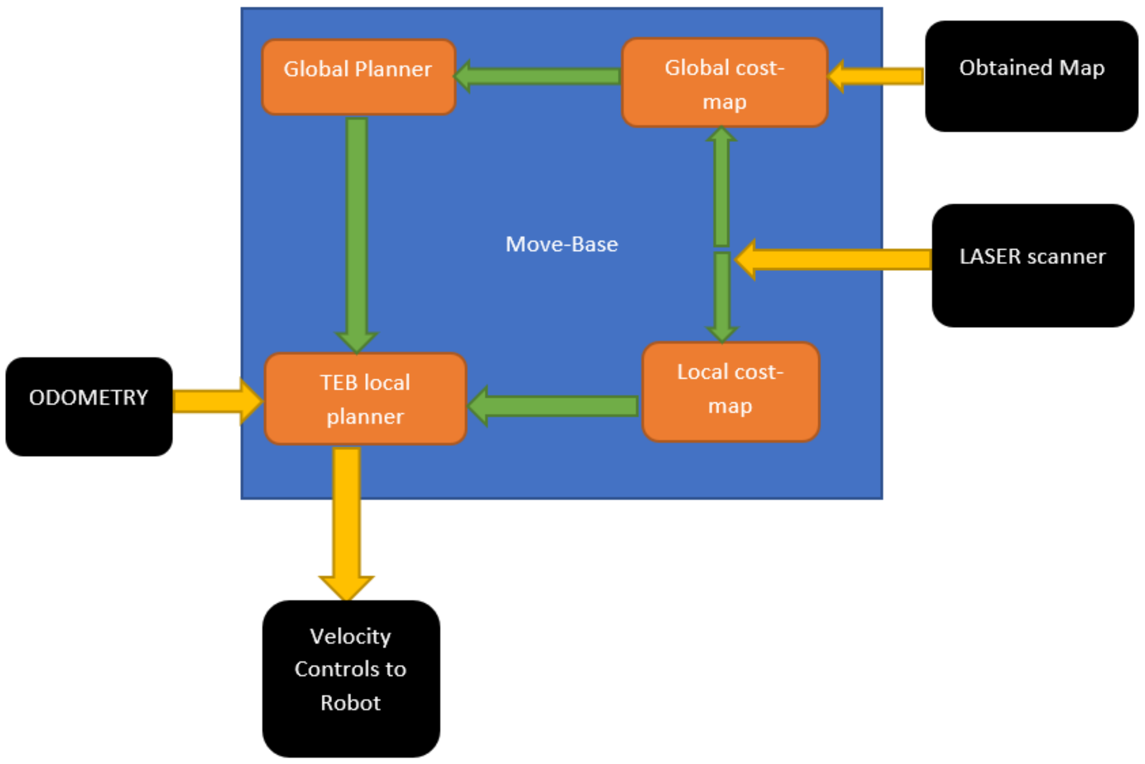


Fig 3.1 Block Diagram of navigation

In Fig 3.1, the block diagram of the components involved in navigation process are presented. The obtained map of the virtual or real environment is included in the global cost-map node, which passes the map to the global planner. The global planner generates an initial trajectory by default for the robot to take, which is then modified by the TEB local planner below it, which receives information about unexpected obstacles from odometry data and laser scan data provided by the robot which is first given to the local cost-map and then to the TEB local planner. After this, the TEB local planner gives velocity commands to the robot in order to perform navigation.

3.4 Flow Chart of Process

The methodology that's being used in the project, or the basic operation, is described as follows in the flowchart illustrated in Figure 3.2.

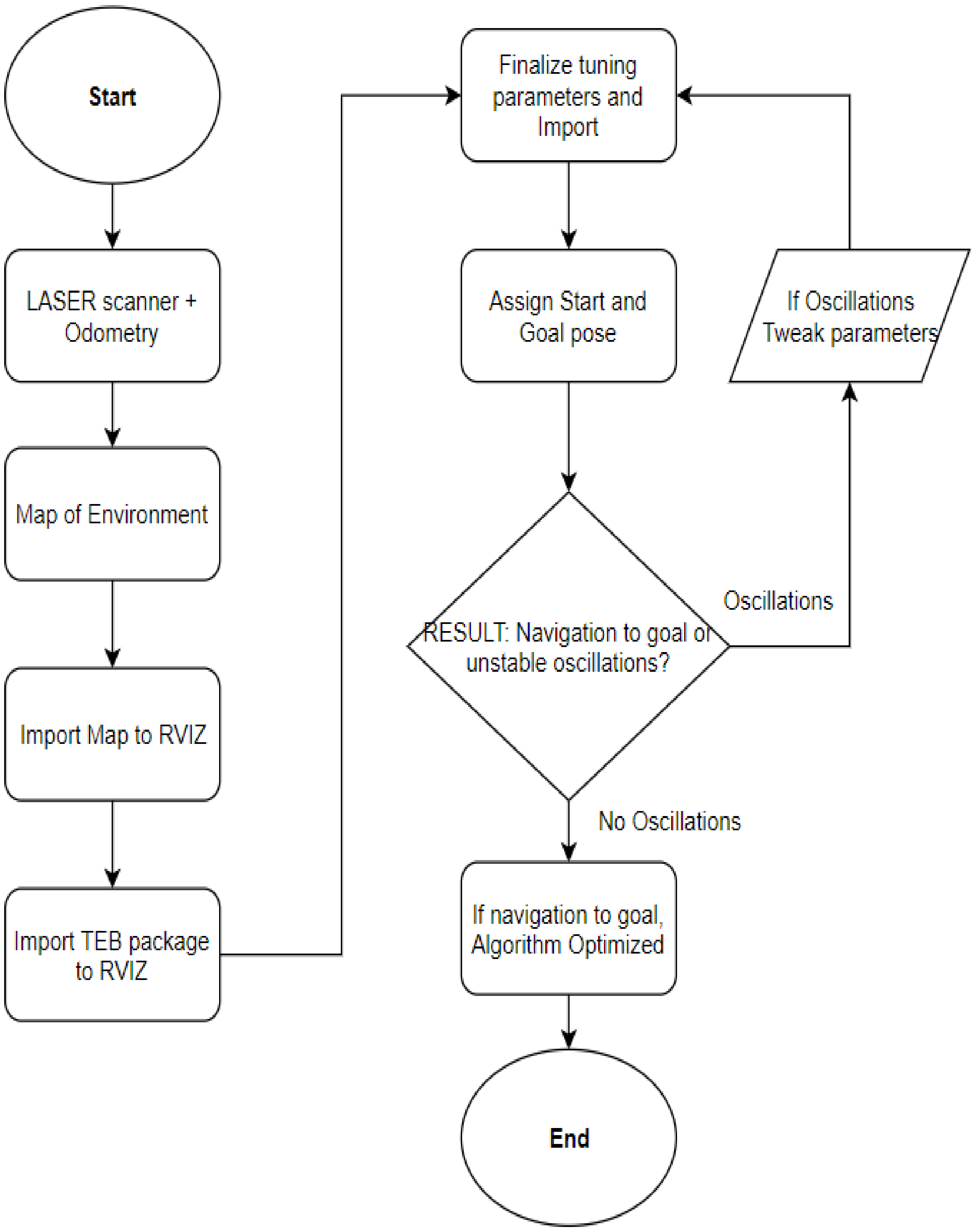

Start
LASER scanner + Odometry
Map of Environment
Import Map to RVIZ
Import TEB package to RVIZ
Finalize tuning parameters and Import
Assign Start and Goal pose
RESULT: Navigation to goal or unstable oscillations?
Oscillations
If Oscillations Tweak parameters
No Oscillations
If navigation to goal, Algorithm Optimized
End

Fig 3.2: Flowchart describing the methodology of the process

In figure 3.2, initially, a 2D map of the environment of the robot is obtained by manually controlling the robot to explore its navigation space, and the LASER scan data along with odometry information “perceives” the world around the robot in two-dimensions, thus forming a map that is as accurate as possible. The map of the virtual or the real environment can be visualized on another tool called the RVIZ simulator.

After this is done, in the mean-time, the TEB local planner package is imported into the move-base launch file as said before, which contains the variable tuning parameters. The parameters need to be finalized and then the algorithm needs to be tested, which is done by assigning an initial and a final pose to the robot in the map, and the result is noticed.

If the robot successfully reaches the assigned goal pose, the algorithm is optimized and the navigation is successful. If not, the variable parameters need to be tweaked and adjusted, and the process is repeated again. This is repeated until the newly tweaked parameters result in successful navigation of the mobile robot in the indoor environment.

3.5 Summary

In this chapter, the methodology that describes the basic operation used was discussed. The design approach was presented which talks about the components and tools used in order to implement and realize the proposed methodology.

The block diagram describing the individual elements of the navigation was presented and the role of each element was explained and discussed.

A flowchart that explains the methodology of the basic operation and the steps needed to be done to perform navigation optimization was illustrated and discussed.

CHAPTER 4

TECHNICAL SPECIFICATIONS

This Chapter presents the software and hardware tools used in this project to carry out indoor robot navigation.

4.1 ROS Platform

Robot Operating System (ROS) is a middleware for robotics (i.e. group of software frameworks for robot research and application development). Although ROS itself is not an OS, it provides facilities developed for a varied computer cluster such as abstraction of hardware, low-level hardware control, application of commonly used functions, inter-process communication, and management of packages. Executing sets of ROS-based processes are represented in a graphical structure where processing takes place in nodes that may subscribe, publish and glean sensor data, control, path plan, actuation, and inter-node communication. ROS platform has many versions such as ROS Hydro, ROS Indigo, ROS Jade, ROS Kinetic, ROS Lunar and ROS Melodic. In this project, however, ROS Melodic is being used since it's the most recent and up to date version of ROS.

4.2 TEB local planner package library

The TEB package implements an online trajectory planner which is optimal and local for control and navigation of robot motion which can be included the ROS navigation library [40]. The global planner gives an initial trajectory which is modified during runtime (real-time) based on reducing the execution time of the trajectory (reach goal in less time), collision prevention and adherence to kino-dynamic constraints such as velocity bounds and acceleration limits. The current application favors the kinematics of robots that are non-holonomic (dual wheeled and four wheeled robots). In order to get an optimal trajectory, a multi-objective function is solved which is sparse and weight-based, as mentioned in section 2.3.2. The optimizer can be provided weights and these weights can be adjusted in order to address the algorithm's behavior in case of problems with the objectives [41].

4.3 Software Tools

The following are the software tools used for performing simulation before implementation on an actual robot in a real indoor environment.

4.3.1 GAZEBO

Gazebo is a powerful 3D simulation software for robotic systems that is particularly suitable for testing obstacle-avoidance and computer vision. Gazebo has the ability to utilize numerous high-performance physics engines, such as ODE, Bullet, etc. (the default is ODE). It provides realistic representation of environments including realistic lighting, shadows, graphics and three-dimensional shapes. It can model virtual sensors that perceive the virtual environment, such as laser range scanners, cameras (including wide range), Kinect style sensors, etc.

Gazebo provides a set of virtual environment templates which can be used to perform navigation such as a house, a grid, an empty space, etc [39]. For example, two templates, a hexagonal grid and an indoor house like environment, are shown in Figures 4.1 and 4.2.

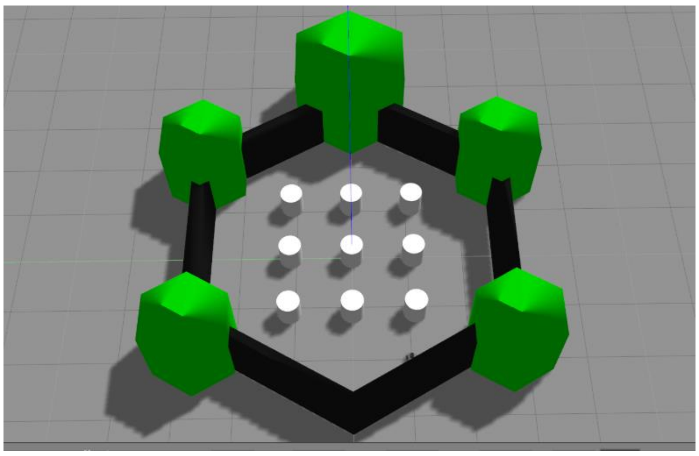


Fig 4.1 – Template 1 of GAZEBO environment

In figure 4.1, one of the templates of the Gazebo environment is presented. It’s a hexagonal grid in which the circular regions represent the obstacles. The robot can move about in the region restricted by the boundaries of the hexagonal space, and can be controlled autonomously as well as manually, depending on the positions of the obstacles.

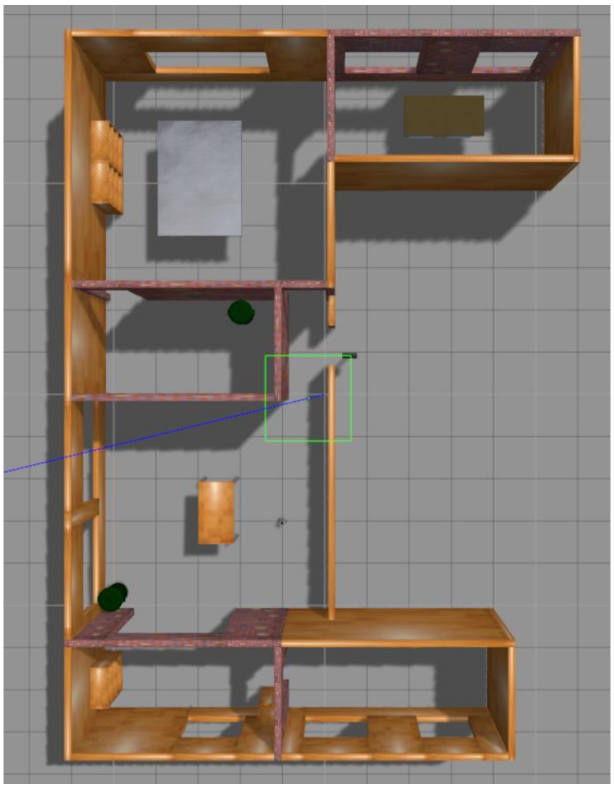


Fig 4.2: Template 2 of the GAZEBO environment

In Figure 4.2, an indoor house-like environment is the template of the GAZEBO simulator, fitted with obstacles like chairs, walls, tables, etc. There is a door like space for the robot to move out of the house here. Similar to template 1, the robot can move about the boundary regions and can be controlled autonomously as well as manually.

The aforementioned images are two of the navigation related templates presented. The user can choose either one as desired and can use them in order to work on various applications.

4.3.2 RVIZ

RVIZ stands for ROS Visualization. As the name says, it’s a 3D visualization tool for ROS which can visualize sensor data as well as the state information as recorded by the robot. Real-time feed of sensor data coming over topics of ROS such as infrared distance values, camera images, sonar values, and much more can be displayed onto this simulator. In addition to this, the map of the indoor environment can also be visualized, which is recorded by on-board scanners of the robot.

4.3.3 TURTLEBOT 3

TURTLEBOT is a standard robot for ROS. TurtleBot3 is a small, low-cost, programmable, ROS-based mobile robot for use in studying, research, hobby, and product prototyping. The aim of TurtleBot3 is to drastically simplify the size of the platform and reduce the cost without having to lose its quality, features and functionality, while simultaneously offering expandability.

There are three releases of the TURTLEBOT. Tully of Open Robotics and Melonee of Fetch Robotics from Willow Garage developed Release 1 on top of the iRobot’s Roomba-based research robot, Create, for use in ROS. It was developed in 2010 and has been on sale since 2011. In 2012, Yujin Robot released the second version based on the iClebo Kobuki research robot. In 2017, Release 3 was developed with features that were not present in its predecessors, and also in the users’ demands. The third version deploys ROBOTIS smart actuator Dynamixel for motion. The TURTLEBOT 3 is also available as a virtual robot which can be used on the GAZEBO platform for various applications. The TURTLEBOT is illustrated in Fig 4.3 [37].

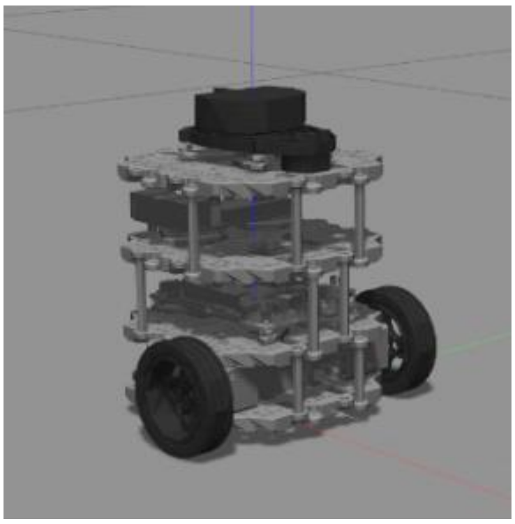


Fig 4.3 The TURTLEBOT 3 Burger

TURTLEBOT3 has three models: Burger, Waffle and Waffle-pi. Fig 4.3 shows the TURTLEBOT 3-Burger version. The robot is modified with low-cost and an embedded system made up of a Single Board Computer, 360-degree distance sensor and also 3D printing. TURTLEBOT is available as an open-source package library for ROS, which includes all three versions of the robots. Since the robot is available for simulation as well, these technical aspects can be exploited free of cost.[37]

4.4 List of Hardware

After testing the navigation on the simulator, it is done again on the hardware, using a real robot. Figure 4.4 shows the actual robot on which navigation algorithm is tested.

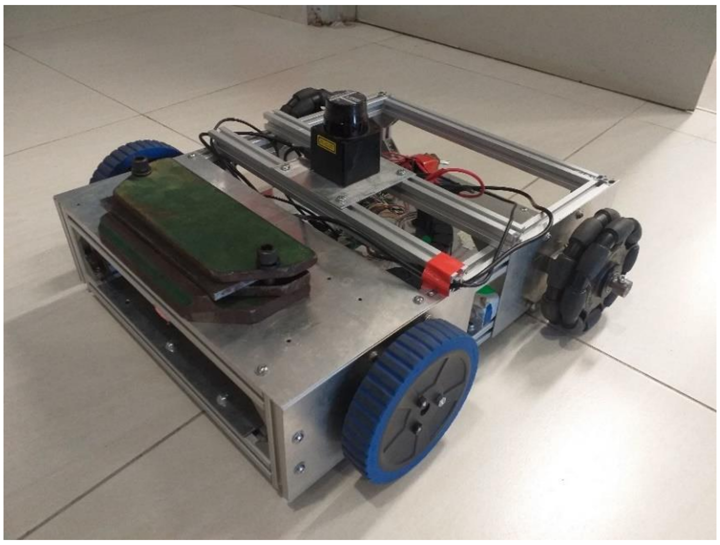


Fig 4.4 Robot used for real-time navigation

Figure 4.4 presents the four-wheeled robot on which the navigation was tested. It has many hardware components used for building it, which are discussed as follows.

- Hokuyo URG-04LX-UG01 LASER scanner to obtain map of the environment
- DC motors for wheel motion
- 4 Wheels: two for driving and two omnidirectional wheels
- dsPIC33FJ128MC804 – PIC based Microcontroller prototype board
- Li-Ion battery for powering up the controller.
- Connecting Wires.
- Some weights to place on the robot for creating friction

4.5 Summary

Thus, in this chapter, the technical specifications used for the project were discussed. The need to use ROS as the main platform was explained, and the teb_local_planner package was used in order to incorporate the navigation algorithm onto the ROS platform. The tools used for software, GAZEBO, RVIZ and TURTLEBOT and the list of hardware were presented and explained.

CHAPTER 5

IMPLEMENTATION OF NAVIGATION ALGORITHM

This Chapter presents the steps followed for performing the implementation of the navigation algorithm.

5.1 The Navigation Environments

In order to perform navigation, the robot needs a navigation space. Navigation is first performed on a virtual environment before doing it on a real environment since there is no risk of malfunctioning on the virtual robot, unlike the real robot. Thus, in order to test the prototype of the robot, the environments chosen are a hexagonal space as a virtual space and an indoor corridor as the actual space. The chosen environments are presented in Figures 5.1 and 5.2.

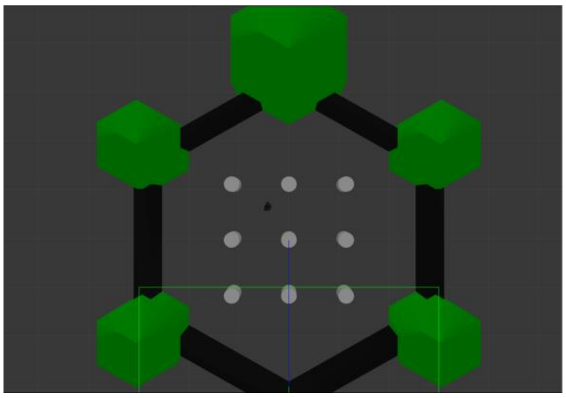


Fig 5.1: The virtual hexagonal environment

In Fig 5.1, the environment chosen for performing virtual navigation is presented. It’s a hexagonal space bounded at the edges, and the robot can move about only in this space. The circular regions depict the obstacles. This specific template was chosen as it is simple and feasible in order to test and implement the prototype.

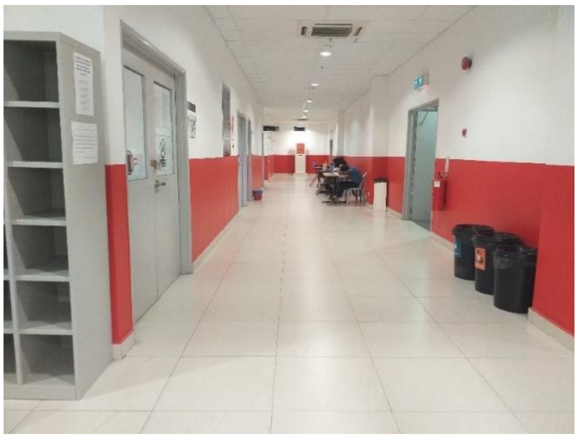


Fig 5.2: The environment chosen for real-time navigation

Fig 5.2 shows the environment used for actual navigation, which is a T-shaped corridor of a building, about five meters wide in cross-sectional width. The corridor was made of racks, doors, chairs and tables which were the obstacles for the moving robot.

5.2 Obtaining the maps of the environments

The map of the environment is crucial for the robot as it needs to localize itself and be aware of its position. In order to obtain the map of an environment, the robot is first manually controlled in order to explore the entire space of the environment. While exploring, the robot uses a LASER scanner to perceive obstacles and boundary limits of the navigation space and uses odometry data to construct the map. For the user to assign start and goal positions to the robot, the user needs to assign it using the obtained map.

Manual control is given depending on the type of the environment. For the virtual environment, simple keyboard commands are given for robot motion. For the real environment, the following type of control is given, shown in Fig 5.3.

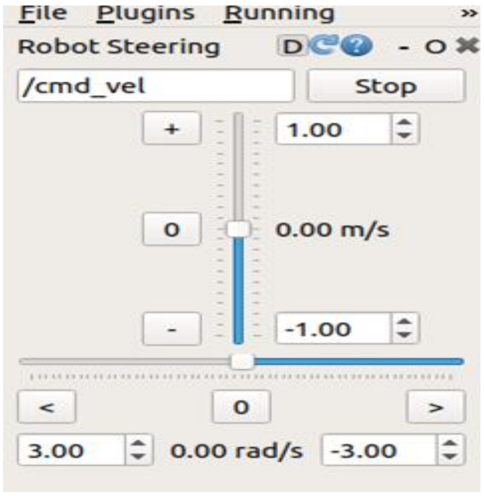


Fig 5.3: Control panel for real robot

In Fig 5.3, the method of controlling the real robot is illustrated. It's a system for giving velocity commands to the robot. The vertical lever is used to give translational velocity in meters/sec (forward and backward), and the velocity in radians/sec is the rotational velocity (If positive, turn left else turn right).

The commands has to be given from the PC to the on-board PIC microcontroller, and this cannot be done wirelessly; the PC would need to move along with the robot, and this would be a tedious task. Therefore, a second PC was used which was connected to the former (which is placed on the robot), through SSH (Secure Shell) and a LAN connection using the building's WIFI. In this manner, the on-board PC's controlling terminal is mirrored into the second PC. The user then controls the robot with the second PC to explore the corridor environment, using the controls given in Fig 5.3.

5.3 Tuning Parameters and their variations

As discussed before, the TEB algorithm can be implemented on ROS by means of an open source package library, the teb_local_planner package [41]. This library package allows us to adjust values in order to change the algorithm's performance. These values are classified into different categories: robot settings, goal tolerance, obstacle constraints, trajectory settings, optimization constraints, and also planning in multiple topologies.

The parameters at their default values work well for the robot that's built by the developers of the TEB package. It may not necessarily work in a similar fashion for other robots, including the robots being used in the project. Therefore, it is important to understand the meaning of each parameter and tweak them until the performance is optimized and the package implements the algorithm well on the desired robot.

The following sections illustrate the different parameters that form the basis of the TEB package library and their meanings in tabular columns [41], as well as the changes needed to be done in order to customize and improve the performance of the algorithm on the robot. Many parameters can be modified at runtime, and the resulting changes in the performance of the algorithm can be observed by the user.

5.3.1 Robot Settings

Robot settings parameters pertain to the robot's physical dimensions, the shape of the structure, and the velocity and acceleration limits both in terms of translational motion (forward and backward) as well as rotational motion (turning clockwise and counter-clockwise).

Table 5.1 Robot Settings Parameters

Name of parameter and default value	Definition and SI Units
acc_lim_x (default value: 0.5)	Maximum value of linear acceleration of the robot in (meters/sec^2)
acc_lim_theta (default value: 0.5)	Maximum value of angular acceleration of the robot in (radians/sec^2)
max_vel_x (default value: 0.4)	Maximum value of linear velocity for the robot (meters/sec)
max_vel_theta (defult value: 0.3)	Maximum value of angular velocity of the robot (radians/sec)
max_vel_x_backwards (default value: 0.2)	Maximum absolute value of linear velocity of the robot while moving backwards (meters/sec)
footprint_model (default value: "point")	The shape and type of the robot used for optimization. Various kinds: "point", "circular", "line", "two_circles" and "polygon." The type of the model drastically affects the required time of computation.(String)
radius (for circular, default value:0.5)	If the robot has a circular shape, this value represents the radius of that circle (meters).

Table 5.1 lists the robot configuration parameters, their definition, their default values and their SI units. These the default values were obtained by the developers of the TEB package library [41]. These values, however, work well for the robot built by the developers and may not necessarily work for other robots. Since the robot being used in the virtual environment is the TURTLEBOT 3 burger, the following changes were made.

- max_vel_theta was increased from **0.3 rad/sec** to **2.8 rad/sec**.
- radius was decreased from **0.5 meters** to **0.1 meters**
- Footprint model was changed from type **“point”** to type **“circular”**

The reason for making these specific changes is described in the documentation of the TURTLEBOT 3 burger [37], where the physical dimensions and the velocity and acceleration limits are mentioned.

5.3.2 Trajectory Settings

This section describes the trajectory settings parameters. These are the variables which correspond to the path which the robot takes, and depends on time-based aspects as well as the discrete poses on the path. Table 5.2 illustrates the trajectory configuration parameters.

Table 5.2: Trajectory Settings Parameters

Name of parameter and default value	Definition and SI Unit
dt_ref (default value: 0.3)	Desired value for the trajectory's temporal resolution (constant time interval between two successive poses) (seconds).
dt_hysteresis (default value: 0.1)	Hysteresis for automatic resizing based on the current resolution of time, generally about. 10% of dt_ref is a desirable value (No unit) [41]
global_plan_overwrite_orientation (default value: true)	Overwrite orientation of local subgoals provided by the global planner.
max_global_plan_lookahead_dist (default value: 3.0)	Mention the maximum euclidean length (cumulative distance) of global plan's subset that is considered in order for optimization (meters).
feasibility_check_no_poses (default value: 4)	Number of poses on the outcome path up to which the feasibility of the path is checked with each sampling interval. (no unit)

Table 5.2 illustrates the trajectory related parameters, their default values, their definitions and their SI units. When tested on the TURTLEBOT with the default values, the robot couldn't navigate to the assigned goal, and got stuck due to oscillations. The oscillations are due to poor performance of the algorithm, and convergence didn't happen in order to reach the optimum solution. Thus, in order to overcome this problem, the following changes were made.

- The parameter dt_ref was slightly increased from **0.3** to **0.5** seconds in order to speed up optimization and ensure convergence of the algorithm.
- The parameter feasibility_check_no_poses was increased from **4** to **8** to check whether more upcoming poses were affected by obstacles or not. The value was set to 8, since for values greater than 8, the algorithm couldn't converge properly.

These specific changes were made to the parameters in order to improve the algorithm's performance and ensure convergence of the algorithm [41].

5.3.3 Obstacle Constraints

The obstacle constraints pertain to the presence of obstructions and obstacles in the path of the moving robot. The parameters are based on separation from a particular pose of the robot, the poses which the robot needs to consider whether an obstacle is there or not, etc. Table 5.3 describes the obstacle parameters for the TEB algorithm [41].

Table 5.3 Obstacle Constraints

Name of parameter and default value	**Definition and SI Unit**
Min_obstacle_dist (default value: 0.4)	Minimal distance between robot and obstacles (meters).
Include_costmap_obstacles (default value: true)	Whether obstacles in the robot's vicinity should be considered. (Boolean value)
Costmap_obstacles_behind_robot_dist (default value: 1.0)	Restrict the present obstacles within range that are considered to be planned behind the robot (meters).
Obstacle_poses_affected (default value: 30)	The number of poses in the trajectory which are near an obstacle and whether the robot should consider those poses to be affected by obstacle or not. (No Unit)

Table 5.3 illustrates the obstacle parameters, their default values, SI units and their definitions. At the default values, the robot oscillates at a particular position and doesn't move forward. Therefore, the following changes were made.

The following are the changes made to the obstacle parameters

- The min_obstacle_distance was significantly decreased from **0.5 meters** to **0.15 meters**. This was done because the TURTLEBOT 3 burger had a radius of 0.1 meters, meaning a diameter of 0.2 meters. Due to the small size, the obstacle separation had to be given a value less than 0.2 meters and greater than 0.1 meters
- The obstacle_poses_affected was reduced from **30** to **15**, again due to the relatively small size of the TURTLEBOT 3 burger. Values between 10 and 20 are suitable for a robot of the given dimensions [37].

Thus, the above changes were made due to the small size of the TURTLEBOT, and due to the documentations given in [37] and [41].

5.3.4 The Optimization constraints

Optimization weights pertain to the objective function expressed by equation (2), given in section 2.3.1. The parameters pertain to the weights of the objective function, and are illustrated as follows in Table 5.4.

Table 5.4 Optimization constraints

Name of parameter and default value	Definition and SI Unit
no_inner_iterations (default value: 5)	Number of inner iterations of the optimizer called by the outer loop (No Unit)
no_outer_iterations (default value: 4)	Number of outer loop iterations performed by the outer loop optimizer for modifying the trajectory based on the value of dt_ref and then calling the inner loop iterations (No Unit)
weight_max_vel_x (default value: 2)	Weight of optimization for ensuring the value of maximum feasible linear velocity(No Unit)
weight_max_vel_theta (default value: 1)	Weight of optimization for ensuring the maximum possible rotational velocity (No Unit)
weight_acc_lim_x (default value: 1)	Weight of optimization for ensuring the maximum possible linear acceleration (No Unit)

weight_acc_lim_theta (default value: 1)	Weight of optimization for ensuring the maximum possible angular acceleration (No Unit)
weight_kinematics_turning_radius (default value: 1)	Weight of optimization for establishing a minimum turning radius (No Unit)
weight_obstacle (default value: 50)	Weight of optimization for having a least distance from obstacles (No Unit)

Table 5.4 illustrates the Optimization parameters. The values of these parameters are usually small, since the they are weights of the objective cost function given by equation (2), discussed in section 2.3.1. However, the following changes had to be made to the optimization parameters.

- Weight_kinematics_turning_radius had to be decreased from **1** to **0.6**. This value had to be less than 1 because the TURTLEBOT 3 operates with two wheels i.e. it's a differential drive robot[37].
- Weight_obstacle had to be decreased from **50** to **35** due to the small size of the TURTLEBOT 3 burger, and since the min_obstacle_dist parameter had to be decreased in section 5.3.3.

The aforementioned changes were made to optimize the algorithm's behavior [41], and to ensure convergence of the optimizer.

5.3.5 The Goal Tolerance Limits

This section lists the goal tolerance limits, which are limits pertaining to all the goal-pose conditions. These parameters of goal-tolerance are listed in table 5.5.

Table 5.5: Goal Tolerance Limits

Name of parameter and default value	Definition and SI Unit
xy_goal_tolerance (default value: 0.2)	Allowable max. euclidean distance to the goal point (meters).
yaw_goal_tolerance (default value: 0.2)	Allowable max. error of orientation (radians)
free_goal_vel (default value: false)	This value must be removed to allow the robot to get to the goal with maximum speed (Boolean value)

Table 5.5 illustrates the goal tolerance parameters, their SI Units, definition and default values [41]. In this section, the only change that was made was to the xy_goal_tolerance parameter, which was decreased from **0.2** meters to **0.1** meters.

The reason for this is the small size of the TURTLEBOT 3 burger with a radius of 0.1 meters. With the xy_goal_tolerance value of 0.1 meters, it would mean that the robot is as close to the goal as possible [41].

5.3.6 Parameters of Parallel Planning in Distinctive Topologies

This section describes the parameters involved in choosing alternative trajectories. The parameters are based on the concept of homotopy class as discussed in section 2.3.2. Table 5.6 lists the parameters pertaining to this category.

Table 5.6: Parameters of Parallel planning in distinctive topologies

Name of parameter and default value	Definition and SI Unit
enable_homotopy_class_planning (default value: true)	To allow planning for alternate trajectories in different topologies (Boolean Value)
max_number_classes (default value: 4)	The maximum number of different trajectories that can be considered (No Unit)
h_signature_prescaler (default value: 1.0)	Scaling of H-signature variable that is used to distinguish between homotopy classes (No Unit)
enable_multithreading (default value: true)	Turn on multiple threading so that the trajectories can be decided in different threads (Boolean Value)
roadmap_graph_area_width (default value: 6)	Width of the rectangular region between the start point and goal point containing random way points (meters)

Table 5.6 illustrates the parameters pertaining to planning alternative trajectories of a homotopy class. The algorithm with the default values may not work well when the number of obstacles near a particular pose of the robot is too high. Thus, in order to address the problem, the following changes were made.

Changes made to the parameters are as follows:

- H_signature_prescaler was reduced slightly from **1** to **0.8**. This was done because with the default value, there were issues with numerous obstacles present in the local cost map. Also, the value 0.8 was given because a lower value would mean that obstacles cannot be distinguished from each other [41].
- Max_number_of_classes was increased from **4** to **6**. This is because this number depends on the number of obstacles in the navigation space. A bigger number means that the optimizer has more options of trajectories to choose. However, this value should not be too high, as it would be computationally expensive [41].

5.4 Summary

In this chapter, the implementation i.e. the steps involved in using the TEB algorithm on a mobile robot for navigation were discussed. The methodology used to obtain the map of the indoor environment was discussed, and the underlying tuning parameters of the TEB local planner package were listed in tabular columns along with their definitions and SI units.

Certain changes were made to certain parameters in order to make the performance of the algorithm better, and the parameters that were changed were discussed along with their updated values, and the reason for updating them was also mentioned.

CHAPTER 6

RESULTS AND DISCUSSIONS

This chapter compiles the results obtained after implementing the navigation algorithm on the virtual, and real environment.

6.1 Map of the virtual environment

As discussed in section 5.2, manual controls were given to the robot to explore the whole navigation space, and obtain the map of the environment by means of laser scan data. Keyboard commands were given for the robot in the virtual environment, the map of which can be seen in Figure 6.1.

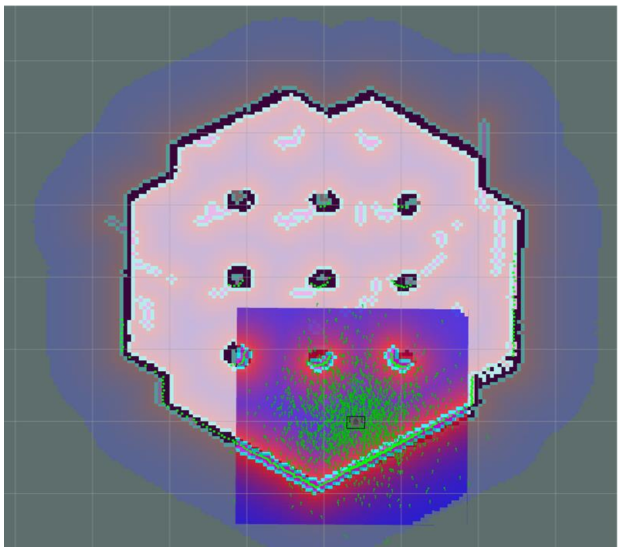


Fig 6.1 Map of the Virtual environment

In Fig 6.1, the map of the virtual hexagonal environment provided by gazebo, is visualized on the RVIZ tool. The map is obtained using laser scan and odometry. The black circles denote the circular obstacles seen in the GAZEBO template, and the dark square region is the coverable region of the robot's laser scanner. The virtual robot, TURTLEBOT 3 is at the center of this square region. The start/goal pose can be assigned by the user on the map.

6.2 Navigation in the Virtual Environment

Now that the map of the virtual environment is available, the navigation has to be performed. In order to do this, the robot in the map has to be assigned a start and a goal pose, which is done on the RVIZ simulation tool. Figures 6.2a, 6.2b, 6.2c and 6.2d illustrate the navigation operation of the mobile robot. Fig 6.2a shows the initial condition of the robot after the RVIZ

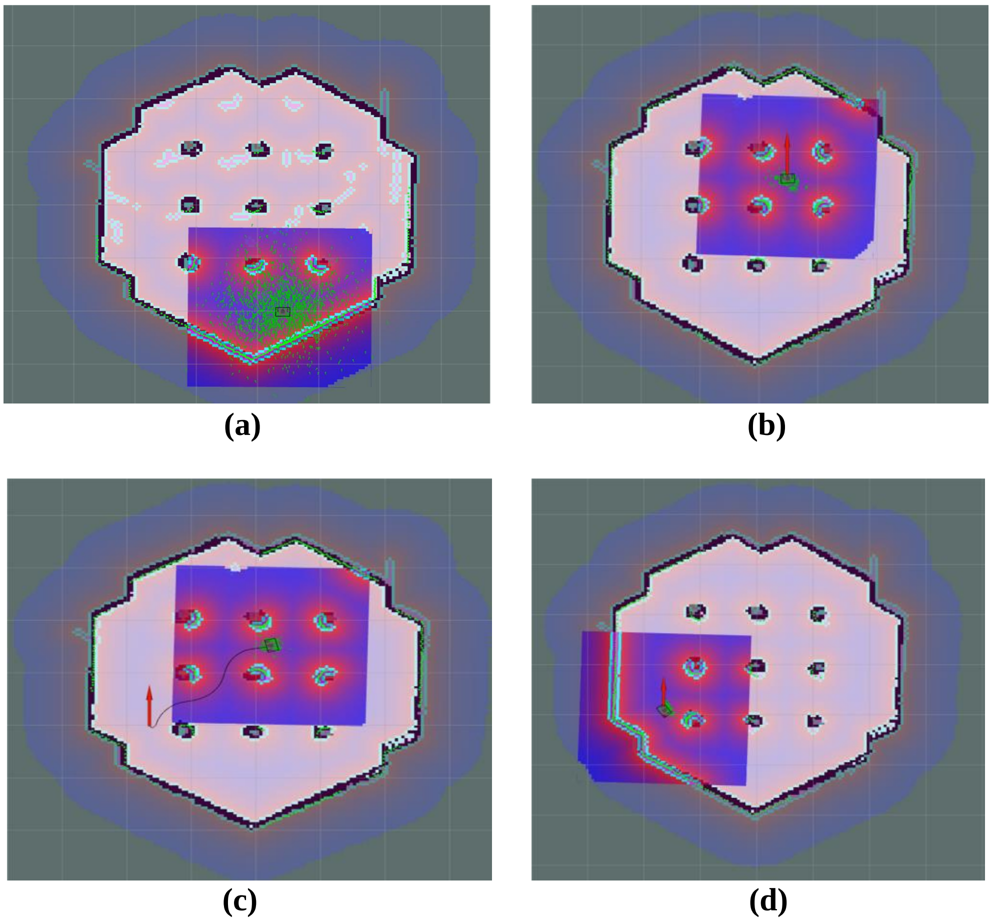


Fig 6.2. **(a)** Initial Condition **(b)** Assign Start pose **(c)** Assign Goal pose **(d)** Reach goal

tool has been opened by default. In Fig 6.2b, the start pose of the robot has been assigned, and the red arrow denotes the direction faced by the robot. In Fig 6.2c, the goal pose has been assigned, and the robot begins navigation in order to reach the goal. The curvy black line connecting the start and the goal pose is the path taken by the robot. And finally, in Fig 6.2d, it can be seen that the robot has reached its assigned goal, and the red arrow is the direction it is facing, which is known when the goal was assigned in the beginning.

6.3 Map of the Real Environment

Similar to the case of the virtual environment, the map of the corridor as discussed in section 5.1 was obtained by manually controlling the four-wheeled robot within the corridor using the velocity commands for forward, backward, left and right rotate as discussed in section 5.1, and also by means of the LASER scanner and odometry data. The map of the indoor corridor, which is a T-shaped environment, is presented in figure 6.3 as follows.

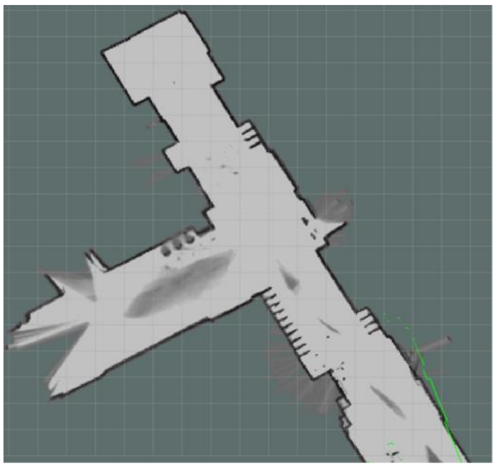
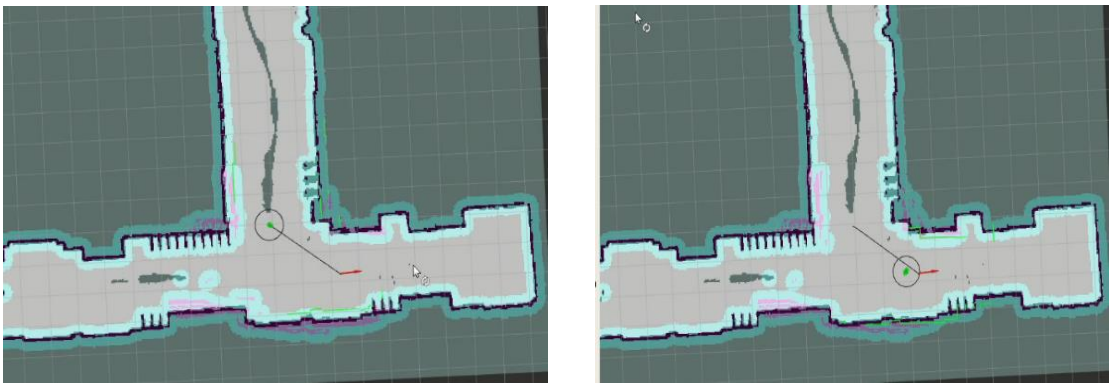


Fig 6.3 Map of the indoor corridor

The map of the indoor corridor environment is illustrated in Figure 6.3. This map is also visualized on the RVIZ tool, and is obtained by the URG LASER scanner present on the four-wheeled robot, as discussed in section 4.5. The boundary of the region is the black envelope throughout the corridor region. Similar to the virtual case, the start and goal pose is assigned by the user on this map, and navigation takes place.

6.4 Navigation in the real-environment

In the map of the corridor environment, the navigation is performed by assigning a start and a goal pose, and the user can assign these poses as desired. Figures 6.4a and 6.4b illustrate the navigation of the mobile robot in the indoor corridor.



(a) **(b)**

Fig 6.4 (a) *Robot at start pose* **(b)** *Robot at goal pose*

In Figure 6.4a, the robot is assigned the start pose. The circular region at the start pose denotes the space in the environment that is occupied by the robot, which is of a radius of 0.5 meters. Figure 6.4b was taken when the robot reached its assigned goal pose. Like the virtual case, the black line represents the trajectory taken by the robot. In this case, where no obstacles were present, the trajectory is a perfectly straight line, which is the fastest path to the goal.

6.5 Workflow of the process

The following image shows the workflow of different nodes that collaborate together to perform the task of navigation, represented by a rqt graph [41]. Each node is mentioned in a bubble, and the arrows represent the flow of control of the entire process.

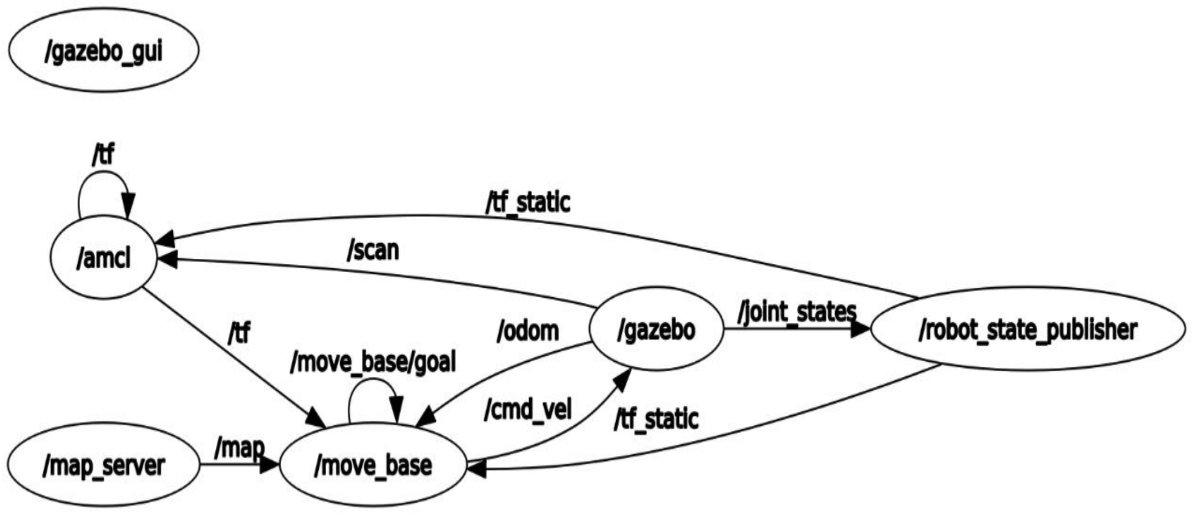


Fig 6.5: RQT graph showing workflow of navigation algorithm

In Fig 6.5, the workflow of the algorithm is represented as a RQT-graph. An RQT graph is a graph which represents the work flow and co-ordination between multiple components called nodes, as a state-space diagram. Each bubble represents what is called a node. A node is an individual segment of a process which is assigned a particular task, and multiple nodes each assigned with a particular task co-ordinate with each other and work together to perform a larger overall operation.

In the rqt-graph, the gazebo_gui node is responsible for generating the virtual environment template provided by the gazebo platform, and the AMCL node launches the code which performs localization for the robot in an environment, using sensors and interactions with the environment.

The move_base node contains the teb_local_planner library plug-in and launches it in order for the algorithm to run. It also contains the global planner for initiating a default trajectory, global and local cost-maps for getting the information about the map of the environment as well as a map of the obstacles near the robot, as discussed in section 3.3.

The gazebo node represents the virtual environment in which the robot is present, and passes on the control commands given by the move_base node to the robot_state_publisher. The map-server node provides the map initially obtained by the robot using laser scan data. The robot state publisher gives us the state of the robot in the virtual environment gazebo.

6.6 Summary

Thus, in this section, the following information was discussed.

- Obtaining the map of the virtual and real environment by means of manual control and laser scan data
- The maps were illustrated on the RVIZ simulation tool
- Navigation in the virtual and real environment by assigning desired start and goal poses on the 2D map
- Illustration of the Workflow of the navigation process by means of what is called an RQT graph
- Explanation of the nodes that constitute the structure of the RQT graph and thus the basis of the navigation process.

CHAPTER 7

CONCLUSION AND FUTURE WORK

This chapter concludes the thesis on Indoor navigation algorithm optimization sand suggests additional functionalities for possible future work and implications.

7.1 Conclusion

In this project, the optimization of a robot navigation algorithm using ROS (Robot Operating System) was demonstrated. An analysis of the numerical parameters involved in the functioning of the algorithm was done. After careful tweaking, the parameters that affect the performance of the robot and the trajectory were observed and recorded. Tweaking of parameters was done during simulation until optimum performance was achieved and the newly updated parameters were recorded. The parameters and the velocity control commands were then given to an actual robot and the final setup was tested and run in real-time, in a real indoor corridor environment.

7.2 Future Work and Implications

In this project, the TEB algorithm was implemented with static obstacles in the environment. The aim in future is to apply the algorithm in an environment where there are dynamic (moving) obstacles and customize the behavior according to this situation.

With the help of open source libraries like the aforementioned TEB package, enthusiasts and researchers can use them to work on more sophisticated mobile robotic systems such as Autonomous cars, delivery robots, wheel-chair robots, surveillance robots, etc. With the right tuning of parameters and proper tools for simulation and hardware, any kind of mobile robotic device can be made and customized as per the user’s desires.

CHAPTER 8

REFERENCES

[1] S. Quinlan, “Real time modification based on collision free paths”, Stanford University, Stanford, CA, USA, 1995

[2] Christoph Rösmann, Frank Hoffmann, Torsten Bertram, ‘Integrated online trajectory planning and optimization in distinctive topologies’, International Journal of Electrical, Computer, Energetic, Electronic and Communication Engineering Vol:7, No:12, 2013

[3] C. Rösmann, W. Feiten, T. Wösch, F. Hoffmann and T. Bertram, “Trajectory modification considering dynamic constraints of autonomous robots”, Proc. 7th German Conference on Robotics, Germany, Munich, 2012, pp 74–79.

[4] Francisco A. X. Da Mota et al., Localization and Navigation for Autonomous Mobile Robots Using Petri Nets in Indoor Environments, 2018 IEEE Access, Volume 6.

[5] Chih-Chung Chou, Feng-Li Lian, Chieh-Chih Wang, “Characterizing Indoor Environment for Robot Navigation Using Velocity Space Approach With Region Analysis and Look-Ahead Verification”, IEEE Transactions on Instrumentation and Measurement (Volume: 60, Issue: 2, Feb. 2011)

[6] L. Jaillet and T. Simeon, “Path deformation roadmaps: Compact graphs with useful cycles for motion planning,” The International Journal of Robotics Research, vol. 27, no. 11-12, pp. 1175–1188, 2008.

[7] J. Mattingley, Y. Wang, S. Boyd, ”Receding Horizon Control: Automatic Generation of High-Speed Solvers”, in IEEE Control Systems Magazine, Vol. 31, No. 3, pp. 52-65, 2011.

[8] R.A. Knepper, S. Srinivasa, M.T. Mason, “Toward a deeper understanding of motion alternatives via an equivalence relation on local paths”, Int. J. Robot. Res. 31 (2) (2012) 168–187.

[9] Woojin Chung et al, “Safe Navigation of a Mobile Robot Considering Visibility of Environment”, IEEE Transactions on Industrial Electronics (Volume: 56 , Issue: 10 , Oct. 2009)

[10] Mohamed M. Atia ; Shifei Liu ; Heba Nematallah ; Tashfeen B. Karamat ; Aboelmagd Noureldin, “Integrated Indoor Navigation System for Ground Vehicles With Automatic 3-D Alignment and Position Initialization”, IEEE Transactions on Vehicular Technology (Volume: 64 , Issue: 4 , April 2015)

[11] Hongtai Cheng, Heping Chen ,Yong Liu, “Topological Indoor Localization and Navigation for Autonomous Mobile Robot”, IEEE Transactions on Automation Science and Engineering (Volume: 12 , Issue: 2 , April 2015)

[12] Hung-Yuan Chung , Chun-Cheng Hou , Yu-Shan Chen,” Indoor Intelligent Mobile Robot Localization Using Fuzzy Compensation and Kalman Filter to Fuse the Data of Gyroscope and Magnetometer”, IEEE Transactions on Industrial Electronics (Volume: 62 , Issue: 10 , Oct. 2015)

[13] Dora Luz Almanza Ojeda, Yazmin Gomar Vera, Mario Alberto Ibarra Manzano, “Obstacle Detection and Avoidance by a Mobile Robot Using Probabilistic Models”, IEEE Latin America Transactions (Volume: 13 , Issue: 1 , Jan. 2015)

[14] Marija Seder, Mato Baotić, Ivan Petrović, “Receding Horizon Control for Convergent Navigation of a Differential Drive Mobile Robot”, IEEE Transactions on Control Systems Technology (Volume: 25 , Issue: 2 , March 2017)

[15] Seong Jin Kim, Byung Kook Kim, “Dynamic Ultrasonic Hybrid Localization System for Indoor Mobile Robots”, IEEE Transactions on Industrial Electronics (Volume: 60 , Issue: 10 , Oct. 2013)

[16] S. Bhattacharya, “Topological and geometric techniques in graph-search based robot planning”, Ph.D. dissertation, University of Pennsylvania, January 2012

[17] D. Lee, W. Chung, “Discrete-Status-Based Localization for Indoor Service Robots”, IEEE Transactions on Industrial Electronics (Volume: 53 , Issue: 5 , Oct. 2006)

[18] Sunhong Park, Shuji Hashimoto, “Autonomous Mobile Robot Navigation Using Passive RFID in Indoor Environment”, IEEE Transactions on Industrial Electronics (Volume: 56 , Issue: 7 , July 2009)

[19] L. Vachhani, Arun D. Mahindrakar, K. Sridharan, “Mobile Robot Navigation Through a Hardware-Efficient Implementation for Control-Law-Based Construction of Generalized Voronoi Diagram”, IEEE/ASME Transactions on Mechatronics (Volume: 16 , Issue: 6 , Dec. 2011)

[20] Zhe Huang, Jigui Zhu, Linghui Yang, Bin Xue, Jun Wu, Ziyue Zhao “Accurate 3-D Position and Orientation Method for Indoor Mobile Robot Navigation Based on Photoelectric Scanning”, IEEE Transactions on Instrumentation and Measurement (Volume: 64 , Issue: 9 , Sept. 2015)

[21] M. Segura, V. Mut, C. Sisterna “Ultra-wideband indoor navigation system”, IET Radar, Sonar & Navigation (Volume: 6 , Issue: 5 , June 2012)

[22] Ren C. Luo, Tung Jung Hsiao “Dynamic Wireless Indoor Localization Incorporating With an Autonomous Mobile Robot Based on an Adaptive Signal Model Fingerprinting Approach”, IEEE Transactions on Industrial Electronics (Volume: 66 , Issue: 3 , March 2019)

[23] Ana Corrales Paredes, Maria Malfaz, Miguel A. Salichs, “Signage System for the Navigation of Autonomous Robots in Indoor Environments”, IEEE Transactions on Industrial Informatics (Volume: 10 , Issue: 1 , Feb. 2014)

[24] P. Rusu, E.M. Petriu, T.E. Whalen, A. Cornell, H.J.W. Spoelder, “Behavior-based neuro-fuzzy controller for mobile robot navigation”, IEEE Transactions on Instrumentation and Measurement (Volume: 52 , Issue: 4 , Aug. 2003)

[25] Chun-Han Lin, Chung-Ta King, “Sensor-Deployment Strategies for Indoor Robot Navigation”, IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans (Volume: 40 , Issue: 2 , March 2010)

[26] S.Y.P. Chien ; L.Q. Xue ; M. Palakal, “Task planning for a mobile robot in an indoor environment using object-oriented domain information”, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) (Volume: 27 , Issue: 6 , Dec 1997)

[27] L. E. Kavraki et al.”Probabilistic roadmaps for path planning in high-dimensional configuration spaces”, in IEEE Transactions on Robotics and Automation, Vol. 12, No.4, pp.566-580, August 1996.

[28] D. Fox, W. Burgard, and S. Thrun, “The dynamic window approach to collision avoidance,” IEEE Robotics & Automation Magazine, vol. 4, no. 1, pp. 23–33, March 1997.

[29] Fiorini, P.; Shiller, Z. Motion planning in dynamic environments using velocity obstacles, International Journal on Robotics Research, Vol. 17, No. 7, July 1998, pp. 760-772

[30] S. M. LaValle. “Planning Algorithms”, Cambridge University Press, Cambridge, U.K., 2006.

[31] Reeds, J.A.; Shepp L.A. Optimal paths for a car that goes both forwards and backwards, Pacific Journal of Mathematics, Vol. 145, No. 2, 1990, pp. 367-393

[32] Graf, B.; Wandosell, J.M.H.; Schaeffer, C,.”Flexible Path Planning for Nonholonomic Mobile Robots”, Fraunhofer Institute Manufacturing Engineering and Automation (IPA), 2001

[33] S. Quinlan, O. Khatib, “Elastic bands: connecting path planning and control”, [1993] Proceedings IEEE International Conference on Robotics and Automation

[34] F. Lamiraux, D. Bonnafous, O. Lefebvre, ”Reactive path deformation for nonholonomic mobile robots”, in IEEE Transactions on Robotics, Vol. 20, No. 6, pp. 967-977, 2004.

[35] C. Rösmann, W. Feiten, T. Wösch, F. Hoffmann and T. Bertram, “Efficient trajectory optimization using a sparse model”, Proc. IEEE European Conference on Mobile Robots, Spain, Barcelona, 2013, pp. 138–143.

[36] C. Rösmann, F. Hoffmann and T. Bertram, “Planning of Multiple Robot Trajectories in Distinctive Topologies”, Proc. IEEE European Conference on Mobile Robots, UK, Lincoln, Sept. 2015

[37] http://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/#simulation

[38] Kaiyu Zheng, “ROS Navigation Tuning Guide”, arXiv:1706.09068 [cs.RO]

[39] https://www.generationrobots.com/blog/en/robotic-simulation-scenarios-with-gazebo-and-ros/

[40] http://wiki.ros.org/

[41] C. Rösmann, “teb_local_planner ROS Package [Online]”, 2015. URL http://wiki.ros.org/teb_local_planner.

CHAPTER 9

BIODATA

Name	: P. S. N. SURYAVAMSI
Phone Number	: +91-9176069261
E-mail	: ps.nsuryavamsi2015@vit.ac.in
Permanent Address	: Gems Park Apts, Mogappair West, Chennai-600037