저작자표시-비영리-변경금지 2.0 대한민국

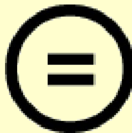
이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

조선대학교
CHOSUN UNIVERSITY

August 2022

Master's Degree Thesis

Real-Time Map-Based Autonomous Navigation of Mobile Robot Using ROS

Graduate School of Chosun University

Department of Electronic Engineering

Henok Tegegn Warku

조선대학교
CHOSUN UNIVERSITY

Real-Time Map-Based Autonomous Navigation of Mobile Robot Using ROS

ROS를 사용하는 이동 로봇의 지도 기반 실시간 자율 주행

August 26, 2022

Graduate School of Chosun University

Department of Electronic Engineering

Henok Tegegn Warku

조선대학교
CHOSUN UNIVERSITY

Real-Time Map-Based Autonomous Navigation of Mobile Robot Using ROS

Advisor: Prof. Nak Yong Ko

This thesis is submitted to Graduate School of Chosun University in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In

Electronics Engineering

April 2022

Graduate School of Chosun University

Department of Electronic Engineering

Henok Tegegn Warku

조선대학교
CHOSUN UNIVERSITY

This is to certify that the Master's Thesis of Henok Tegegn Warku

Has been approved by the Examining Committee for the thesis requirement for the Master's Degree in Electronic Engineering

Committee:

Prof. Hong Gi Yeom, Chosun University (sign)

(Chair Committee)

Prof. Nak Yong Ko, Chosun University (sign)

(Committee Member)

Prof. Sung Hyun You, Chosun University (sign)

(Committee Member)

May 2022

Graduate School of Chosun University

조선대학교
CHOSUN UNIVERSITY

TABLE OF CONTENTS

LIST OF FIGURES....................iii

LIST OF TABLES....................viii

ABSTRACT....................x

한 글 요 약....................xi

1. INTRODUCTION....................1

1.1. Research Background and Paper Reviews....................1
1.2. Objectives....................3
1.3. Organization of Thesis....................4

2. SOFTWARE PLATFORM....................5

2.1. Software Platform....................5
2.1.1. Robot Operating System....................5
2.1.2. ROS Simulation Environment....................8

3. ALGORITHMS FOR NAVIGATION OF MOBILE ROBOT....................10

3.1. Probability Theory....................10
3.2. Monte Carlo Localization....................12
3.3. Occupancy Grid Mapping....................14
3.4. Simultaneous Localization and Mapping Algorithm....................15
3.4.3. Feature-Based FastSLAM....................16
3.4.4. Grid-Based FastSLAM....................17
3.5. Path Planning....................18
3.5.1. Global Planner....................19
3.5.2. Local Planner....................20

4. SIMULATION AND HARDWARE SETUPS....................24

4.1. Simulation Setup....................24
4.1.1. Model of Robot....................24
4.1.2. Interfacing of Simulated Robot and Lidar Sensor....................24

조선대학교
CHOSUN UNIVERSITY

4.1.3. Simulation Environment used for Navigation of Mobile Robot 26
4.2. Hardware Setup 27
4.2.1. Robot Platform 27
4.2.2. Sensors on Scout Mini 29
4.3. Structure of ROS Navigation 31
4.3.1. SLAM-gmapping 37
4.3.2. Adaptive Monte Carlo Localization 40
4.3.3. Move-Base Package 44
5. RESULTS AND DISCUSSIONS 53
5.1. Simulation Results 53
5.2. Experimental Results 64
5.3. Development of Graphical User Interface 83
5.4. GUI-Based Autonomous Waypoint Navigation of Mobile Robot 84
5.5. Effects of Parameters on the Mobile Robot Performance 89
6. CONCLUSION 101
REFERENCES 102
PUBLICATIONS 105
ACKNOWLEDGEMENTS 106

조선대학교
CHOSUN UNIVERSITY

LIST OF FIGURES

Figure 2-1 ROS architecture block diagram 6

Figure 2-2 ROS Gazebo Simulator. 8

Figure 2-3 ROS Visualization (Rviz) tool. 9

Figure 3-1 Visualization of Adaptive Monte Carlo Localization (AMCL) for different states. (a) shows the particle filter after running Global Localization, where all particles have been spread evenly throughout the map. (b) depicts that the particle filter has converged to low variance. 14

Figure 3-2 Breadth-first expansion in the Dijkstra algorithm 19

Figure 3-3 Best-first expansion in A* Algorithm 20

Figure 3-4 Dynamic-Window Approach (DWA) 22

Figure 4-1 The simulated platform of the robot on Gazebo and Rviz from left to right. 24

Figure 4-2 The URDF of Hokuyo lidar sensor and mobile robot in a simulated environment (a) URDF of Hokuyo lidar sensor on Rviz. (b) Mounting the lidar sensor on the top of the robot using URDF on the Gazebo simulator. (c) Facing the wall (the light brown color) as an obstacle in front of the robot to observe the lidar output. The blue color is the ray of the lidar activating Gazebo's ray visualization plugin.(d) Visualizing the internal state of the laser scan (the red color shows the detected obstacle in front of the robot from (c)) on Rviz. 25

Figure 4-3 URDF representation of Hokuyo lidar sensor. 26

Figure 4-4 Transformation tree diagram of the mobile robotSimulation Environment used for Navigation of Mobile Robot. 26

Figure 4-5 Simulation Environment. (a) The built mobile robot inside the environment and lidar rays using Gazebo. (b) Rviz's internal lidar signal visualization by detecting obstacles in the environment. 27

Figure 4-6 The hardware platform of the mobile robot used for experimentation. (a) Robot Hardware setup, (b) Mobile robot on Rviz, and (c) Frame transformation on Rviz. 28

Figure 4-7 External topology connection of Scout Mini. 29

조선대학교
CHOSUN UNIVERSITY

Figure 4-8 Nvidia AGX Xavier external topology connection. 29
Figure 4-9 Intel RealSense D435 camera 30
Figure 4-10 Velodyne VLP-16 lidar sensor. 31
Figure 4-11 Robot process cycle using ROS 32
Figure 4-12 Challenges to Navigation of mobile robot 32
Figure 4-13 ROS Navigation stack structure 34
Figure 4-14 Odometry source information of a mobile robot. (a) Shows the two-wheeled mobile robot that is used for simulation in this thesis (MRP-NRLAB02 Red-One technology). (b) The odometry data from the mobile robot's wheel encoder containing position, orientation, linear velocity, and angular velocity with topic name */odom*, and the data can be visualized with ROS command of *$ rostopic echo /odom*. (c) It shows the type of message that the */odom* topic contains with nav_*msgs/Odometry's* message type and it can be extracted with the ROS command of *$ rostopic info /odom*.(d) Shows the visual representation of odometry information............... 34
Figure 4-15 The transformation between the coordinates of laser and robot base link 35
Figure 4-16 URDF of lidar sensor transformation with the simulated robot on Rviz 36
Figure 4-17 Example of Occupancy Grid Map (OGM). 38
Figure 4-18 The PGM file and the metadata of the YAML file. 38
Figure 4-19 Examples of AMCL 41
Figure 4-20 AMCL node data subscription and publication 41
Figure 4-21 Example of a global planner 45
Figure 4-22 Example of global costmap 48
Figure 4-23 Example of a local planner... 49
Figure 4-24 Example of local costmap. 51
Figure 5-1 Simulation environment constructed on Gazebo. 53
Figure 5-2 Occupancy Grid Map of the virtual environment on Gazebo and Rviz 54
Figure 5-3 Communication between nodes and topics using rqt_graph 56
Figure 5-4 Transformation tree after launching the mapping algorithm. 56

조선대학교
CHOSUN UNIVERSITY

Figure 5-5 Preparing the simulated mobile robot for navigation on Rviz (a) and Gazebo (b) on the static obstacles in the environment. 58

Figure 5-6 Correction of the robot's pose for navigation on Rviz (a) and Gazebo (b) following a small robot motion 59

Figure 5-7 Autonomous navigation of mobile robot in a simulated environment with static obstacles of the environment. 60

Figure 5-8 Communication of node and topic of the navigation system. 61

Figure 5-9 Dynamic obstacles in the simulated environment. 62

Figure 5-10 Autonomous navigation of a mobile robot in a simulated environment with dynamic obstacles of the environment. 63

Figure 5-11 The target position and orientation (a) and the estimated pose (b) of the robot in Figure 5.10 64

Figure 5-12 Experimental indoor environments for navigation of a mobile robot 65

Figure 5-13 Mapping of the actual environments for Scenarios 1 (a) and 2 (b), with comparisons of environmental features and the constructed map on Rviz. 67

Figure 5-14 Block diagram representation of the mapping algorithm of the experiment 68

Figure 5-15 Manual localization of mobile robot for Scenario 1 and Scenario 2 using 2D pose estimator tool (a), (c) The yellow color arrow of the particles used to estimate the robot pose in Scenarios 1 and 2 depicts the uncertainty in robot position at the beginning of AMCL. The green arrow represents the robot's manual pose estimation in the environment using the *2D pose estimator* tool of Rviz. (b), (d) Represents the condensation of particles after the robot translates 1.6 meters in the positive x-direction for Scenario 1 and 3.29 meters forward in Scenario 2. .. 70

Figure 5-16 Global localization of a mobile robot for Scenario 1 and Scenario 2. (a), (c) After calling AMCL's *global localization* service, the particles are distributed uniformly (yellow arrows) across the entire map, with equal guesses about the robot's pose. Because the robot pose estimation and the actual robot poses differ, the lidar measurement (white line) does not match the map. (b), (d) The convergence of particles after the robot moves and when the filter updates

조선대학교
CHOSUN UNIVERSITY

its belief due to the motion, the measurements were projected from the robot's pose point of view. For Scenario 1 the distance the filter took to converge is 4.15 meters and 21.38 meters for Scenario 2. 71
Figure 5-17 Particles used for Case-2 parameters. 72
Figure 5-18 The effect of small particles on the localization of the robot. 72
Figure 5-19 Structure of particle filter. 73
Figure 5-20 Autonomous navigation of a mobile robot in the actual indoor environment of the static map without dynamic obstacles for Scenario 1 and Scenario 2. 76
Figure 5-21 Global planner of the robot in the static map of the environment. 76
Figure 5-22 Dynamic obstacles on Rviz and the actual environment from left to right for Scenario 1 (a) and Scenario 2 (b). 78
Figure 5-23 Autonomous navigation of a mobile robot in the actual indoor environment of the static map with dynamic obstacles for Scenario 1 and Scenario 2. 80
Figure 5-24 Qt 5 Designer tool. 84
Figure 5-25 The designed GUI for autonomous navigation of a mobile robot through different waypoints(a), (b): GUI platforms that contain the actual environments and the map of the environments on the right side, and also contain 5 tools for controlling the navigation system for Scenario (a) and Scenario (b). 85
Figure 5-26 GUI-based autonomous navigation of a mobile robot for Scenario 1. (a): shows the mobile robot's real-time autonomous navigation to the first waypoint (1) on the map using the GUI platform. (b): depicts the mobile robot's real-time autonomous navigation to the second waypoint (2) of Table 5-8 Scenario 1. (c): shows the mobile robot's real-time autonomous navigation to the third waypoint (3) on the map using the GUI platform. (d): shows the mobile robot's real-time autonomous navigation to the map's initial pose (0) using the GUI platform. 89
Figure 5-27 Robot path for Scenario-1environment. (a) Robot inside the environment., (b) The path of the robot with its position values. 90
Figure 5-28 Effects of map resolution at the start of gmapping. 91

조선대학교
CHOSUN UNIVERSITY

Figure 5-29 AMCL pose estimation for the three particles interval. 93

Figure 5-30 The pose estimates of the robot for the three-particle intervals on Rviz. The figures depict how the particle clouds are dispersed with the lidar matching the environment. (a) min =500, max = 5000 , (b) min = 100, max = 5000, (c) min = 5, max = 20.................................. 94

Figure 5-31 The effect of changing the value of the minimum linear and angular distance to perform filter updates. (a) update_min_d = 0.1, update_min_a = 0.1., (b) update_min_d = 0.03, update_min_a = 0.03., (c) update_min_d = 0.25, update_min_a = 0.2., (d) update_min_d = 0.75, update_min_a = 0.7. 96

Figure 5-32 The effects of the odometry noise and the filter update parameters adjustments and comparisons. (a) case 1, (b) case 2, and (c) case 3 98

조선대학교
CHOSUN UNIVERSITY

LIST OF TABLES

Table 4-1 Hardware specification. 28

Table 4-2 Hardware specification of Intel RealSense D435 camera. 30

Table 4-3 Hardware specification of VLP-16. 31

Table 4-4 Major parameters of gmapping 39

Table 4-5 Major parameters of AMCL. 42

Table 4-6 Move base parameters 44

Table 4-7 Global planner parameters 46

Table 4-8 Global costmap parameters. 48

Table 4-9 DWA planner parameters 49

Table 4-10 Local costmap and common costmap parameters 52

Table 5-1 Some positions and orientations of the mobile robot during the mapping operation. 55

Table 5-2 Comparison of target goal pose and robot estimated pose with their respective errors of static obstacles in the environment ready for autonomous navigation. 60

Table 5-3 Some gmapping parameter values used for the experiment 68

Table 5-4 Some of the AMCL parameters for the two cases 69

Table 5-5 Experimental comparisons of the target goal pose and the real robot estimated pose of Scenario 1 (a) and Scenario 2 (b) of Figure 5.20. 77

Table 5-6 Experimental comparisons of the target goal pose and the real robot estimated pose of Scenario 1 (a) and Scenario 2 (b) for the dynamic obstacle. 81

Table 5-7 Some of the navigation parameters used during the experiment of Scenario 1 and Scenario 2. (a) Global planner params, (b) Global costmap params, (c) DWA local planner params, (d) local costmap params, and (e) common costmap params 82

Table 5-8 Position and attitude quantitative values of the waypoints for autonomous navigation using GUI of Figure 5.25. (a) Scenario 1 and (b) Scenario 2 87

조선대학교
CHOSUN UNIVERSITY

Table 5-9 The effects of map resolution tuning on the time it took the robot to fully explore the environment. 91

Table 5-10 The odometry noise and filter update parameter values. 99

조선대학교
CHOSUN UNIVERSITY

ABSTRACT

Real-Time Map-Based Autonomous Navigation of Mobile Robot Using ROS

Henok Tegegn Warku

Advisor: Prof. Nak Yong Ko, PhD.

Department of Electronic Engineering

Graduate School of Chosun University

Many applications of mobile robotics necessitate the safe planning of a collision-free motion to a defined place. Real-time obstacle avoidance strategies enable reactive motion in dynamic and unpredictable situations, whereas planning approaches are best suited for achieving a goal position in known static environments. A ROS-based approach is presented to address the challenge of robot SLAM, which has been used in real-time applications to construct the map the environment, localize the robot within the environment, plan paths, and avoid obstacles. It is demonstrated that all navigation modules can coexist and work together to reach the destination without colliding with static and dynamic obstacles. The goal of this thesis is to provide a platform for guiding a mobile robot in a real-world environment while avoiding static and dynamic obstacles. Our results were validated at Chosun University in South Korea, through simulation and testing in indoor environments, and the effect of mapping and localization parameters was studied and investigated for better performance of the robot while navigating autonomously in the environment. In addition, a Graphical User Interface (GUI) is being developed to guide the mobile robot through various waypoints autonomously.

조선대학교
CHOSUN UNIVERSITY

한 글 요 약

ROS를 사용하는 이동 로봇의 지도 기반 실시간 자율 주행

Henok Tegegn Warku.
Advisor: Prof. Nak Yong Ko, PhD.
Department of Electronic Engineering
Graduate School of Chosun University

모바일 로봇 공학 분야에서는 모바일 로봇이 정해진 장소로 주행하기 위해 충돌 없는 안전한 주행이 필요하다. 충돌 없는 안전한 주행을 위한 방법으로는 실시간 장애물 회피 방법과 경로 계획 방법이 있다. 먼저, 실시간 장애물 회피 방법은 로봇이 움직이거나 예측할 수 없는 환경에서 주행하기에 적합하고, 경로 계획 방법은 로봇이 고정된 환경에서 목표 위치에 도착하기에 적합하며 이러한 방법들은 ROS 를 통해 구현되었다. ROS 는 항법 모듈인 주변 환경 맵핑, 로봇의 위치 인식, 경로 계획, 그리고 장애물 회피를 위해 모바일 로봇 분야에서 실시간으로 로봇 SLAM 을 구현하는 데 사용된다. 이 논문은 모든 항법 모듈을 함께 작동하여 고정되어 있거나 움직이는 장애물을 충돌하지 않게 회피하면서 목적지에 도달할 수 있도록 실제 환경에서 모바일 로봇을 주행하기 위한 플랫폼을 제공하는 것을 목적으로 한다. 이 논문의 결과는 대한민국에 있는 조선대학교의 실내 환경에서 시뮬레이션과 실험을 통해 검증되었으며, 주변 환경을 자율적으로 탐색하면서 로봇의 더 나은 성능을 위해 맵핑 및 위치 인식 매개변수에 따른 효과를 연구하고 조사했다. 마지막으로, 다양한 경유점을 통해 모바일 로봇을 자율적으로 주행하기 위한 GUI(Graphical User Interface)를 개발하고 있다.

조선대학교
CHOSUN UNIVERSITY

1. INTRODUCTION

1.1. Research Background and Paper Reviews

Mobile robotics has recently become more popular due to technological advancement, the availability of many robot platforms, and robotic system architectures. Numerous autonomous robots have been developed and used for different application areas. Some of them are space and ocean exploration, underground mining, underwater exploration, manufacturing industries, and an autonomous self-driving car that can operate alongside pedestrians and cars driven by humans.

When the robots are modeled and designed, they require a software program and code that can perform a specific task. Programmers and software developers typically write a program for a particular robot that is developed. These programs are frequently customized to the design of the robot and are not versatile. Creating a modular design in hardware can be a relatively simple task, but designing software that is flexible could be very difficult [1].

Data sensors are used for gathering information from the environment to be used by the robot. And also, they play a significant role in deploying Simultaneous Localization and Mappings (SLAMs) for the mobile robots to navigate autonomously. These robots have a huge impact and are the next frontier for technologies that can impact societal life, including industrial manufacturing [2], transport [3], and service robots [4]. Recently, several service robots, such as Care-O-bot [5], NAO [6], and KeJia [7], have had a significant influence on the enhancement of quality of life for people. First, the robot has to understand the environment wherein it is navigating for completing the given task. The conventional way of representing maps are topological, metric, and hybrid [8]. The metric map represents the layout of the environment geometrically by using geometric features such as points or grids, planes, or lines. Whereas a topological map uses a graph for modeling the environment to achieve an abstract representation.

조선대학교
CHOSUN UNIVERSITY

Thus, vertices and edges correspond to places and paths respectively. However, to take advantage of both metric and topological map representations, hybrid map amalgamates the high localization performance of matric map and high path planning accuracy of topological map for the improvement of navigation performance.

Metric navigation for small or medium-scale environments is relatively enough based on an occupied grid or hybrid maps. Nonetheless, for using mobile robots in the case of the domestic scene, it does lack semantic information to be ordered by the users conveniently. For instance, geometric coordinates are used to define the goal point, but humans prefer to interact using natural language. As a result, several researchers are working to create semantic maps that include not only the geometric layout but also the concepts of objects or rooms to improve human-robot interactions[9]. SLAM is commonly used to create metric maps since it can build a map of the environment whilst localizing the robot [10]. SLAM can be categorized according to different sensors used for the collection of information into vision-based and laser-based SLAM. SLAM based on the laser is mostly used to generate occupied grid maps. Representative algorithms include Hector SLAM [11], Cartographer [12], GMapping [13], and, Karto SLAM [14] whereas SLAM based on vision chiefly creates feature maps like lines [15, 16], or planes [17, 18] and points [19, 20]. With the rapid development of the uses of mobile robots, the ability of mapping and localization is crucial for autonomous navigation through the environment. Without the knowledge of the current pose and map of the environment, the robot cannot make its own decisions and actions. Simultaneously Localizing and mapping (SLAM) the environment is crucial for answering autonomous navigation problems in a given environment. To achieve autonomous navigation in indoor environments, mobile robots must be able to obtain information from the environment using range sensors (e.g., laser sensor, 3D sensor, ultrasonic sensor) to construct a map of their environment and determine their location [21].

The ROS navigation stack has been tested using a range of ROS-compatible robots. However, it appears that the impact of parameters in packages like Gmapping, AMCL, and move base have

조선대학교
CHOSUN UNIVERSITY

not been thoroughly investigated. Because the navigation stack has so many parameters to adjust to take the robot from one position to the destination, many had a lot of trouble configuring the parameters for using it for autonomous navigation.

This thesis addresses the simulation and experimental implementation of map-based autonomous navigation for a mobile robot in an indoor environment. We also investigate the impact of tuning major parameters of mapping, localization, and path planning, which will be described in the results and discussion sections (Chapter 5). We also integrated a Graphical User Interface (GUI) with the mobile robot to implement waypoint autonomous navigation of a mobile robot in an indoor environment. This GUI can be used with any ROS-based robot and allows anyone to handle the navigation system without having to write complex commands.

We have noticed that many faced a lot of trouble configuring the parameters of the navigation stack and their impact on the real robot and actual environment on the ROS Wiki platform. Due to this, we tested and recorded data for major parameter change and their effects, and we have uploaded the full implementation guidance manual for real-time map-based autonomous navigation of a mobile robot in an indoor environment to our lab website (*https://irlchosun.wixsite.com/nyko*) that may help others to use it as a guide to navigating their robot autonomously.

1.2. **Objectives**

The main objective of this study is:

- Testing and visualizing different sensors like lidar, IMU, and camera.
- 2D and 3D mapping of different environments on simulation and experimental analysis.
- Localization of the mobile robot inside the constructed map.
- Path planning of the robot. This includes:

a) Global path planning and

조선대학교
CHOSUN UNIVERSITY

b) Local path planning

- Obstacle avoidance. This includes

  a) Static obstacle avoidance

  b) Dynamic obstacle avoidance

- Navigation of mobile robot inside an environment from one point to the desired goal point.
- Development of a Graphical User Interface (GUI) application with multiple options for moving the robot to different waypoints and canceling the operation at any time if problems arise.
- The impact of changing the ROS navigation stack parameters on a mobile robot's mapping, localization, and path planning in order to make the robot more resilient when moving autonomously, as well as investigating some of the factors that can affect the robot navigation performance.

1.3. Organization of Thesis

This thesis is organized into six chapters. The first chapter includes the background of the research, a review of related works, and the objective. The second chapter contains an introduction to ROS and simulation environments that are used such as Gazebo and Rviz. The third chapter is dedicated to the theoretical formulations and algorithms that let the robot localize, build, both localize and navigate through the environment including path planning. Simulation, hardware setup, and the ROS navigation structure are presented in chapter four. Chapter five covers simulation and experimental results. Besides, Graphical User Interface (GUI) based autonomously waypoint navigation system is included. Finally, Chapter 6 presents the conclusion and future works.

조선대학교
CHOSUN UNIVERSITY

2. SOFTWARE PLATFORM

2.1. Software Platform

Before the algorithm that has been developed is deployed onto a real robot, verifying the algorithms using simulation is a very crucial step. The robot simulations and applications are made using the Robot Operating System (ROS) which is the subject of this section.

2.1.1. Robot Operating System

It is an open-source platform for programming robots. It is a collection of libraries, tools, and conventions that facilitate the development of robust and scalable robot actions across a wide range of robotics platforms. It is used in both research and commercial applications and provides robot programming capabilities such as high-level programming language support and tools, message passing interface between processes, availability of third-party libraries, community support, extensive tools and simulators, and so forth. Despite these capabilities, there are still areas in which ROS is not advisable or recommended to develop the actual product owing to security and real-time processing problems [22, 23].

2.1.1.1. ROS Concepts

Conceptually, ROS has three levels: filesystem, computational graph, and community levels [24].

i. Filesystem Level

The filesystem-level chiefly contains resources that we encounter in the disk. The software in ROS is organized as a package. A package may include nodes, datasets, ROS-dependent libraries, and others that are well organized together. This provides useful functionality, thereby the software can be reused easily. The package follows a common structure and has subparts: package manifests, metapackages, repositories, executable files, service types, and message types. Meta-packages represent a group of related packages, whereas package

조선대학교
CHOSUN UNIVERSITY

manifests contain additional information about a package like a name, description, version, dependencies, license, and so on. The data structure of a message, which is sent in ROS, is described by its message type. The response and request data structures for services in ROS are defined by service type. Furthermore, repositories are groups of packages that share a similar version control system [25].

i. Computational Graph Level

The communication between two or more programs can be accomplished using socket programming; however, as the number of programs increases, the complexity as well. This inter process communication can be easily handled using ROS. A robot might have many sensors, actuators, and computing units. So, by writing independent programs for handling sensor data and controlling actuators, the exchange of data between programs can be achieved using ROS, which is better than having a single program. The architecture of ROS for communication between two programs, which are represented by node 1 and node 2, is illustrated in Figure 2-1.

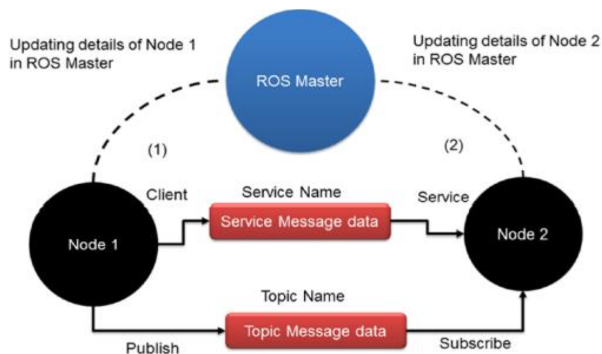


Figure 2-1 ROS architecture block diagram

The communication between nodes is achieved by sending information to the ROS master as well as the data type which is to be sent or received. The nodes can send or receive (publisher and subscriber nodes respectively) different forms of data with each other. The data type of the data (ROS message) can be string, integer, float, and so on. ROS messages are transmitted via a message bus or path known as ROS topics. When a ROS node publishes a topic,

조선대학교
CHOSUN UNIVERSITY

it transmits a ROS topic together with a ROS message, containing data with the message type [22].

ii. Community Level

ROS community-level concepts are resources of ROS that assist various robotics groups throughout the world in sharing their knowledge and programs. These resources are ROS Wiki and answers, ROS distributions, repositories, mailing lists, and blogs. A collection of versioned stacks, a cluster of packages that provide functionality together, can easily be installed from the ROS distributions. To increase the participation of communities, ROS follows a federated repository model rather than having one secured place for all packages, thereby users and developers can create their repositories and also, they are granted the right to control, and license and update their repositories. Moreover, information about ROS is documented in ROS Wiki wherein anyone can sign up and share their documentation, make modifications, develop tutorials, and so forth. Any ROS-related inquiries can also be answered on the ROS answers site [26].

2.1.1.1. Unified Robot Description Format

The Unified Robot Description Format (URDF) is an XML specification that describes the model of the robot. It is created to be as general as possible and makes ROS a modular system. Nodes are made as general as possible for the robot that utilizes them rather than creating unique nodes for different robot types. The robot-specific information that nodes require to do their tasks is contained in the URDF file. A URDF file is constructed in such a way that each link is connected to joints, each robot link is a child of a parent link, and joints are specified by their offset from the parent link's reference frame and rotation axis [27].

2.1.1.2. Coordinate Frames and Transforms

Typically, a robotic system has many 3D coordinate frames, which vary with time, such as base frame, world frame, gripper frame, and so on. Users can use the *tf* package to track

조선대학교
CHOSUN UNIVERSITY

multiple coordinate frames throughout time and keeps track of their relationships in a time-buffering tree structure. In addition to this, it provides the capability for the transformation of points and vectors between any coordinate frame that is chosen at any time [28].

2.1.2. ROS Simulation Environment

i. Gazebo

It is a 3D dynamic simulator used to simulate a robot accurately and efficiently by developing complex indoor and outdoor environments. It provides a more accurate physical simulation of the system, as well as a wider range of sensors with user and program interfaces. Among the basic features of Gazebo are various physics engines, a large library of robot models and environments, and simple programmatic and graphical interfaces [29]. Figure 2-2 shows the ROS Gazebo.

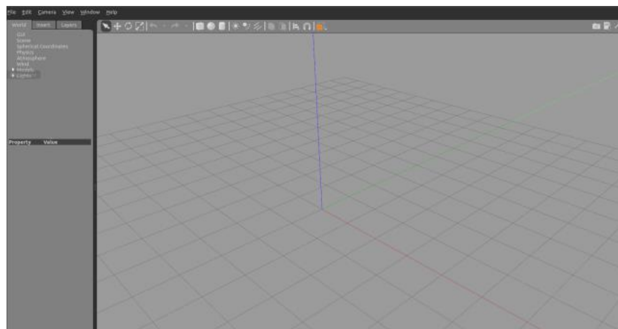


Figure 2-2 ROS Gazebo Simulator.

After designing the robot using Gazebo, the robot's algorithm that is developed using ROS can be tested on the simulated robot. By doing so, model validation can be done that is how the robot performs well as compared to what it was intended to do. Thus, the algorithm will be deployed on the real robot if the predetermined performances have been achieved.

조선대학교
CHOSUN UNIVERSITY

ii. Rviz

ROS Visualization (Rviz) is an impressive 3D visualization tool for ROS that enables users to view or visualize the simulated robot model; that is what is the robot doing, seeing, and heading. It is used to visualize 2D or 3D sensor data from cameras, lidar data as a point cloud, and also a 2D laser range, and webcams as image data [30]. The ROS Rviz is shown in Figure 2-3.

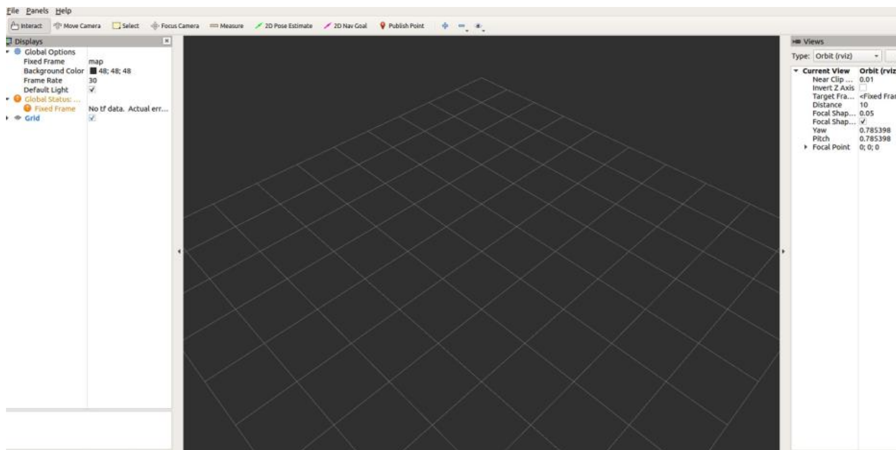


Figure 2-3 ROS Visualization (Rviz) tool.

조선대학교
CHOSUN UNIVERSITY

3. ALGORITHMS FOR NAVIGATION OF MOBILE ROBOT

3.1. Probability Theory

We shall, in this section, commence by introducing the notions and concepts of probability theory that will be used to describe the algorithms listed in this chapter. One might be curious why probability is involved in robotics. This is because robots operate in a real-world that contains numerous uncertainty sources. The interaction with the robot enabled by knowing the data measured by the robot's sensors and the environment is unpredictable. Sensors, on the other hand, have limited resolution and perception due to their range, and are also subjected to noise, which unpredictably disturbs measurements. Since the controls might be noisy and there may be mechanical failures, the motors which drive the robot are not predictable. Finally, the actual information that is required, for instance, the position of the robot on a map, cannot mostly be measured directly and must be inferred. A stochastic model based on probabilistic theory is required to deal with the uncertainties of robot perception and actuation. Most of the explanations of the theoretical concepts and algorithms in this chapter for autonomous navigation of mobile robots are taken from [31]and [32].

The first tool to be introduced deals with the conditional probability, which is the likelihood of an event given the occurrence of another event:

$$p(x|y) = \frac{p(x,y)}{p(y)} \tag{3.1}$$

The Theorem of total probability follows:

$$p(x) = \sum_y p(x,y)p(y) \quad \text{(for discrete)} \tag{3.2}$$

조선대학교
CHOSUN UNIVERSITY

$$p(x) = \int p(x, y)p(y)dy \quad \text{(for continuous)} \tag{3.3}$$

In light of this, the important Bayes' rule on multiple random variables can be determined:

$$p(x|y, z) = \frac{p(y|x, \; z)p(x|z)}{p(y|z)} \tag{3.4}$$

From the above equations, x is the quantity to infer from y, $p(x)$ is the prior probability distribution and y is the data. Other notions used in [32] are shown below:

- Time is discrete, which refers to occurrences that occur at discrete time instants of t = 0, 1, 2, ...
- x_t is the environment at time t and the robot's state. The state can be thought of as the gathering of the environment that can affect the feature (pose, location of the obstacles, the velocity of the robot, and so on) and all properties of the robot.
- z_t is the ***measured data*** time instant *t*, which is knowledge regarding the environment's state. Provided that the measurements between the time interval t_1 and t_2 are taken into account, hence the notation would be $z_{t1:t2}$.
- u_t is ***control data*** which is the corresponding change of the states between time instants *(t-1;t]*. Like the previous case, control data sequences are indicated by $u_{t1:t2}$.

It is essential to note that noise frequently influences both measurement data and control. Because the probabilistic laws govern the measurement and state's evolution, the probability distribution which produces x_t could be written as:

$$p(x_t|x_{0:t-1}, z_{1:t-1}, u_{1:t}) \tag{3.5}$$

However, when the state is complete, which is, knowledge of previous measurements, states, or controls doesn't provide more information to support future predictions, the probability distribution will be:

$$p(x_t|x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(x_t|x_{t-1}, u_t) \tag{3.6}$$

조선대학교
CHOSUN UNIVERSITY

This is because the knowledge of x_{t-1} includes measurement data and controls until the time instant $t-1$.

The other concept in probabilistic theory is *belief*. It is knowledge of the robot regarding the environment. According to the probability theory, beliefs are expressed as a conditional probability distribution, which is the posterior probabilities of the state variables given the measurements and control inputs data:

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t}) \quad (3.7)$$

All of these tools are used to clarify algorithms' mathematical formalization which is used in practice so that the robot creates a map while it localizes itself inside the environment.

3.2. Monte Carlo Localization

The estimation of the robot's pose relative to the known environment of the map is called localization. The 3 localization issues which differ on the available knowledge during the task and initially:

I. **Local localization:** is also known as position tracking and it takes an assumption that the initial pose of the robot is known with small noise. The uncertainty is close to the actual value of the pose of the robot.

II. **Global localization:** the starting pose of the robot is unknown. It is achieved by spreading the particles throughout the map (Figure 3.1. shows Global localization on real robots used for experimentation)

III. **Kidnapped robot problem:** here the robot can vanish and be teleported to a different location.

조선대학교
CHOSUN UNIVERSITY

The MCL algorithm is a particle filter method that uses a set of particles to represent the pose (position and orientation) of the robot. Particle filters are widely used because they are simple to implement and non-parametric, allowing them to represent multimodal probability distributions. Each particle is a separate sample of the robot's state, such as x,y, and heading. Furthermore, each particle has a weighting that indicates how likely it is that a particle represents the actual state of the system. The algorithm works by running the Bayes filter algorithm on each particle. The algorithm has the following steps:

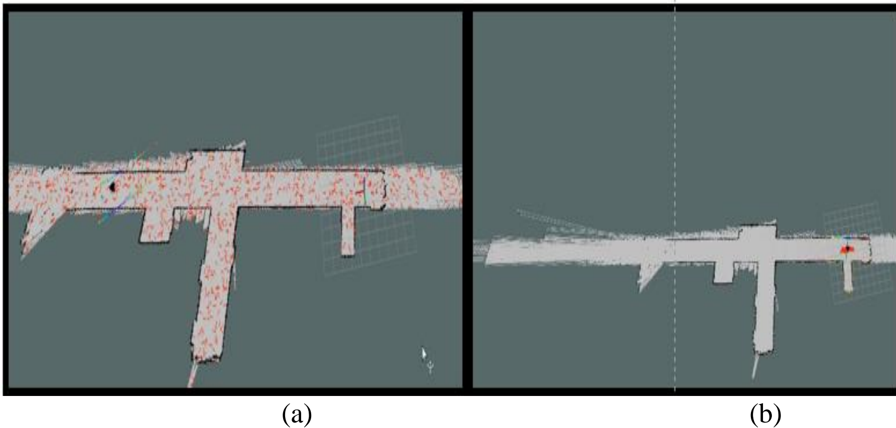
- **Particle sampling:** Calculate the weight from the previous distribution.
- **Motion model:** Updating particle's position using a robot motion model noise.
- **Measurement model:** The given new measurements re-weight particles based on the likelihood that a measurement matches the state of the particle.
- **Resampling:** Replacing unlikely particles of low weight with more likely particles.
- **Compute mean:** Computation of the weighted mean to get the estimated state.

Kalman filter is among the most powerful and important filters used for state estimation in the presence of process disturbance and measurement noise. In mobile robotics, Extended Kalman Filter (EKF) is usually applied to a nonlinear system. The process and measurement models are needed. And additive noises, which are characterized by covariance matrices, are also incorporated in both models. Even though EKF is a robust filter, it has the following drawbacks:

- It depends on the basic principle that is the added noise and the initial belief, $bel(x_0)$ have a Gaussian distribution.
- As the initial belief is required in EKF, the global localization problem can not be addressed using it.
- Feature-dependent maps, for instance, point landmarks are needed for the localization of the robot. The measurements used by the filter for the estimation of the robot's position are represented using the features.

조선대학교
CHOSUN UNIVERSITY

- The filter is found by linearizing the system using Taylor's Expansion, hence it approximates a non-linear system.



(a) (b)

Figure 3-1 Visualization of Adaptive Monte Carlo Localization (AMCL) for different states. (a) shows the particle filter after running Global Localization, where all particles have been spread evenly throughout the map. (b) depicts that the particle filter has converged to low variance.

The Monte Carlo Localization (MCL) can process raw sensor measurements which makes it a great alternative to the EKF. It also avoids the EKF's assumption of uni-modal distribution assumption because of its non-parametric nature. The sensor's range finder is represented using a measurement model algorithm and it returns a probability $p(z_t|x_t, m_t)$ (m represents map) that is the combination of discrete, Gaussian, and exponential distributions. The fundamental idea of MCL is to represent the the $bel(x_t)$ with particles a set of samples that are random states, taken from the belief itself. However, the computational time increases with the number of particles exponentially.

3.3. Occupancy Grid Mapping

Mapping is the way of generating a map of the environment from the noisy measurements provided by the robot's pose. We use mapping algorithms to deal with such problems. This algorithm is based on:

조선대학교
CHOSUN UNIVERSITY

i. **Features:** Recognizing objects within the environment and

ii. **Occupancy grids:** A map can be represented by a grid. Each part of the grid denotes either a free space or an obstacle and a binary value is assigned accordingly.

The posterior over the maps with the given data is:

$$p(m|z_{1:t}, x_{1:t}) \tag{3.8}$$

Where the map is represented by m , $z_{1:t}$ is the set of all measurements up to the time instant t, and $x_{1:t}$ is the sequence of all the poses [13].

3.4. Simultaneous Localization and Mapping Algorithm

It is building a map of the environment that is not known to the robot while localizing itself on the map and only the command inputs $u_{1:t}$ and the measurements $z_{1:t}$ are known by the robot. The robot can build the environment's map while its location is estimated with respect to the map using SLAM. According to the posterior estimate, the problem of SLAM could come in two different forms.

- **Online SLAM**: If the current state and map are estimated given the measurement and control input up to the current time instant and can be expressed as:

$$p(x_t, m|z_{1:t}, u_{1:t}) \tag{3.9}$$

- **Full SLAM**: If the posterior is computed throughout the whole path, $x_{1:t}$, along with the map and it is given by:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}) \tag{3.10}$$

Derivation of online SLAM from full SLAM is by integrating the poses:

$$p(x_t, m|z_{1:t}, u_{1:t}) = \iint \dots \int p(x_{1:t}, m|z_{1:t}, u_{1:t}) dx_1 dx_2 \dots dx_{t-1} \tag{3.11}$$

For convenience, let's take a state vector which is the combination of both the robot pose and the map:

조선대학교
CHOSUN UNIVERSITY

$$y_t = \begin{pmatrix} x_t \\ m \end{pmatrix} \tag{3.12}$$

The posterior (3.17) can be rewritten as:

$$p(y_{1:t}|z_{1:t}, u_{1:t}) \tag{3.13}$$

SLAM based on the Rao-Blackwellized approach uses a combination of particles and Gaussians for representing state variables. A SLAM algorithm that depends on this particle filter version is FastSLAM. In the case of FastSLAM, the robot path is estimated using particle filters so that the errors of each map are conditionally not dependent on each particle. FastSLAM algorithm has the following features:

- It makes use of particle filters to deal with non-linear robot motion models without resorting to linearization.
- Since it computes the whole path posterior, which makes feature locations independent, it addresses both online and full SLAM problems, although it merely estimates one pose at a given time.

FastSLAM algorithm has two major different versions: feature-based and grid-based Fast SLAM.

3.4.3. ***Feature-Based FastSLAM***

This type of FastSLAM algorithm depends on the model of feature-based measurement. In this algorithm, features (f) are extracted from the range measurements, $f(z_t)$, and thus the computational complexity is minimized greatly. It does, however, necessitate the use of additional specialized algorithms for the extraction of features and recognition.

Particles, in feature-based FastSLAM, contain an estimation of pose $z_t^{[k]}$ as well as a collection of Kalman filters for each of the map's m_j features. For estimation of the location of a feature, the Kalman filter is used. The first and second moments are used to characterize it, $\mu_{j,t}^{[k]}$

조선대학교
CHOSUN UNIVERSITY

and $\Sigma_{j,t}^{[k]}$, for the kth particle. Like other particle filter algorithms, it obtains the particle at time instant $t-1$ and the new pose at time t is sampled. It then updates the Extended Kalman Filters (EKF) whenever a new feature is found. Finally, the important weight of the particle is updated, which is used for the resampling process. The representation of the map using feature-based enables the factorization of the posterior (3.17):

$$p(y_{1:t}|z_{1:t},u_{1:t},c_{1:t}) = p(x_{1:t}|z_{1:t},u_{1:t},c_{1:t}) \prod_{n=1}^{N} p(m_n|x_{1:t},z_{1:t},c_{1:t}) \quad (3.14)$$

where $c_{1:t}$ is a variable of the correspondence between the observed feature and the map's genuine feature. It aids in the identification of the observed feature. The underlying principle of the algorithm depends on this factorization since the posterior over robot paths $p(x_{1:t}|z_{1:t},u_{1:t},c_{1:t})$is computed using particle filter whilst each posterior $p(m_n|x_{1:t},z_{1:t},c_{1:t})$ is handled by EKF. The posterior is now factored into N+1 products, but the actual number of filters is MN+1 because each of the M particles is subjected to N Kalman estimations.

3.4.4. *Grid-Based FastSLAM*

The Grid-based FastSLAM algorithm does not rely on EKFs to estimate feature localization because it does not employ a feature-based map; however, it combines MCL with Occupancy Grid Mapping.

The function used for the FastSLAM has the following particular properties:

- ✓ The *sample_motion_model* function computes the sample $x_t^{[k]}$ by integrating the motion model that is the impact of the input u_t on the sample of the previous time instant, $x_{t-1}^{[k]}$; which implies that it computes the posterior $p(x_t|x_{t-1}^{[k]},u_t)$.
- ✓ The *measurement_model_map* function: The significance of the weight w_t of the kth particle is depicted using this function via the probability distribution, $p(z_t|x_t^{[k]},m_{t-1}^{[k]})$

조선대학교
CHOSUN UNIVERSITY

of the measurement z_t given the pose x_t and map m_{t-1} that is computed using the previous measurement and the trajectory followed by the particle.

✓ The *updated_occupancy_grid* function: This function utilizes the pose of the kth particle, the associated map with it, and the measurement for computation of a new occupancy grid.

To summarize, the main benefit of the feature-based FastSLAM approach is that the computational complexity can be controlled. This is achieved by varying both the number N of features and the size of the M set of particles characterizing the map that are localized using EKF. The grid-based technique, on the other hand, leverages particle filters to take advantage of both MCL and Occupancy Grid Mapping. Therefore, a grid-based approach is generally preferred over a feature-based since it does not need features or the use of feature recognition algorithms. Besides, it is more portable since it can model any type of environment.

3.5. Path Planning

Path planning is the problem of creating the path of the robot while navigating from one point to the goal point by avoiding obstacles. The two major tools that are used in path planning are:

✓ A global planner is responsible for determining the best path from a current point to the desired destination.

✓ A local planner tells the robot how to act or handles the command that has to be sent to the robot's wheels for following the global planner.

To put it another way, the global planner seeks the shortest possible path in a known map (a graph or a grid) that is represented digitally whereas the local planner determines how and where the robot should travel. Furthermore, another algorithm is needed to direct the robot which goal to reach to explore the map while the robot is executing autonomous SLAM: frontier

조선대학교
CHOSUN UNIVERSITY

exploration algorithm. Because searching for a goal is one part of path planning, hence frontier exploration is presented [33].

3.5.1. *Global Planner*

This algorithm determines the optimal path the robot has to take from its starting point to its destination. In SLAM, a map of the surroundings is created from the data that is collected using sensors as the robot moves. Using the map that is created, the global planner finds the optimal trajectory by considering the path length and obstacle avoidance. Provided that the endpoint is in an unknown location, the global planner uses the straight line connecting the desired point to its closest possible known point as a trajectory.

Dijkstra and A* are two widely used path planning algorithms in a 2-Dimensional grid map. Both will be covered in the following sections.

Dijkstra Algorithm

In this algorithm, the shortest possible distance is computed in a given path whilst taking into account the lowest cost of the distance between the current points to the target point. It utilizes nodes in its computation and the points where the cost of the distance is low are saved. Figure 3.2 shows operating graphs of the Dijkstra algorithm which is a *breadth-first-search algorithm*. The term "breadth-first" refers to expanding the search in all feasible avenues rather than selecting one in particular; it determines the best or optimal route from a single source vertex to all others [33].

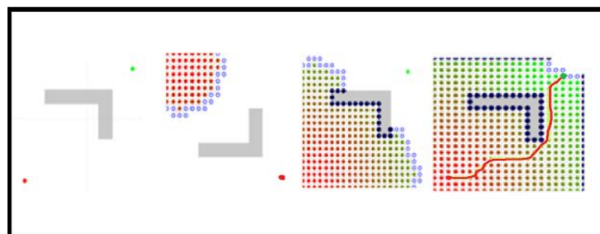


Figure 3-2 Breadth-first expansion in the Dijkstra algorithm

조선대학교
CHOSUN UNIVERSITY

A^* Algorithm

A^* algorithm is also a well-known graph traversal path planning algorithm and it operates in a similar way to Dijkstra's algorithm; however, it directs its searching towards the promising states which saves computational time tremendously. [34]. And also, it is widely used to approach the optimal solution [35] with the data set that is presented. A^* is a best-first search algorithm which is shown in Figure 3.3. Best-first implies the exploration of nodes in a graph in the direction of the most promising vertex, according to a specified rule.

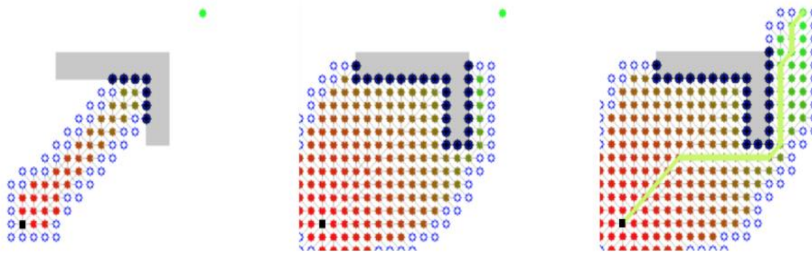


Figure 3-3 Best-first expansion in A* Algorithm

3.5.2. *Local Planner*

It creates new waypoints whilst considering the vehicle constraints and dynamic obstacles to transform the global path into suitable waypoints. Thus, it recomputes the path at a specific rate; the map is scaled down to the vehicle's immediate surroundings and updated as it goes. Since the sensors are not capable of updating the map in all regions, it is impossible to use the whole map and the cost of computation raises with a large number of cells. Therefore, the local planning produces avoidance methods for dynamic obstacles using the updated local map and global waypoints and strives to match the trajectory as closely as feasible to waypoints generated by the global planner [36].

The local planner, from a kinematic standpoint, generates a feasible trajectory from a starting point to the desired location. The starting and target points are the robot's center and a point within a few meters respectively. Using points from a grid-based local cost map, the planner computes a cost function between these two points. The local cost map is a fixed-dimension grid

조선대학교
CHOSUN UNIVERSITY

map that is centered on the robot's (x, y) global map. It calculates the cost of traversing across the grids, taking into consideration the occupancy value of the grids, the robot's velocities, and the distances between the global plan and the target. The information about the cost function is then used by a controller to determine the commanded twist, $([v_x, v_y, \dot{\theta}])^T$ that will be sent to the robot. From the different local planner's Dynamic Widow Approach (DWA) is tested both on simulation and experimental tests.

Trajectory Planner

To begin, it is mandatory to comprehend how Dynamic Window Approach (DWA) works. The standard part [37] of DWA was created for non-holonomic robots that can only have velocities along x and around θ owning to the configuration of their wheel which results in a twist velocities vector of $[v, w]^T$.

DWA executes the following for each iteration:

i. **Search Space:** The space of all possible velocities is calculated by considering the factors cited as follows:

- **Circular Trajectories:** The trajectory that results from the velocity couple (v, w).
- **Admissible Velocities:** By restricting admissible velocities it is ensured that only safe trajectories are taken into account. Speeds are admissible if:

$$v_a = \{(v, w) \mid v \leq \sqrt{2dist(v, w)a_{trans}} \wedge w \leq \sqrt{2dist(v, w)a_{rot}}\} \quad (3.15)$$

where the pair (a_{trans}, a_{rot}) is braking accelerations and dist(v, w) is the distance of the nearest obstacle on a specific (v, w) trajectory.

- **Dynamic Window:** V_d is the dynamic window set and it contains all of the velocities that come from a uniform acceleration motion given the accelerations (a_{trans}, a_{rot}) and the initial velocity (v, w), i.e. the velocity of the robot :

조선대학교
CHOSUN UNIVERSITY

$$V_d = \{(v, w) \mid v \in [v_a - a_{trans}, \mathrm{t}; v_a + a_{trans}, \mathrm{t}] \wedge w \in [w_a - a_{rot}\mathrm{t}; w_a + a_{rot}\mathrm{t}]\} \quad (3.16)$$

The set of all possible velocities is specified with V_s and thus the resulting search space is given by the resulting set V_r as :

$$V_r = V_s \cap V_a \cap V_d \quad (3.17)$$

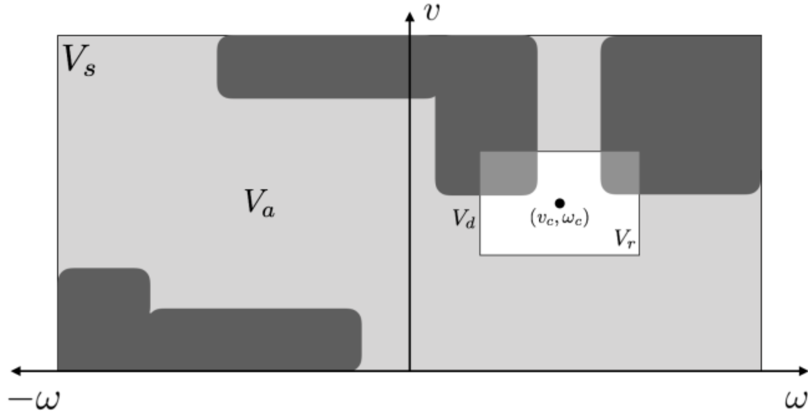


Figure 3-4 Dynamic-Window Approach (DWA)

The above sets can be represented as follows in Figure 3-4: the overall search space with the external rectangle, and the gray region that represents the intersection of the search space and the dark gray areas, which represent the velocities to discard to avoid collisions, represents the set of permissible velocities. The white rectangle represents the Dynamic Window that considers the acceleration; the resulting set V_r is the intersection of the three sets (V_s, V_a and V_d).

- **Optimization:** The distance, heading, and speed for each group of speed pairs in the sampling space of speed is computed by the trajectory evaluation function; it evaluates and chooses the optimal trajectory, and then the robot is driven by the corresponding speed pair. It is given by

$$G(v, w) = \sigma(\alpha \cdot heading(v, w) + \beta \cdot dist(v, w) + \gamma \cdot velocity(v, w)) \quad (3.18)$$

조선대학교
CHOSUN UNIVERSITY

The above function in equation (3.32) is computed for the current pose of the robot and the function trades off the following features:

- **Heading:** The angle between the end of the trajectory related to the speed pair (v, w) and the desired location given by the heading function (v, w). The main purpose of the heading function (v, w) is for selecting a trajectory with a smaller angle to the target position and directing the robot to that location; given by $180^0 - \theta$, where θ is the angle of the target point with respect to the heading direction of the robot.
- **Clearance:** The function $dist(v, w)$ represents the distance between the robot and the obstacle. Its main purpose is to keep the robot from colliding with obstacles.
- **Velocity:** The robot's forward velocity is represented by (v, w) which causes it to move faster to the target position.

The parameters α, β, γ represent the weight coefficients of $heading(v, w)$, $dist(v, w)$, and velocity (v, w) respectively. Normalization is required for the three weighted functions. The weighted sum of the three components is smoothed by function σ, resulting in more obstacle-side clearance.

The trajectory planner [38] is ROS's standard local planner and it is based on the DWA algorithm. It enables us to specify the number of samples that have to be considered for v and w to calculate the *admissible velocities*. And also, whether or not the robot thart is used holonomic.

DWA Local Planner

The DWA local planner utilizes the DWA algorithm as the Trajectory planner; however, the trajectories are computed differently and also, can be applied to holonomic robots [39]. The major parameter *sim_period* parameter, which is the duration of the controller loop, is used for computing Dynamic Window and is inversely proportional to the *controller_frequency* parameter, which will be discussed in the following chapter. The procedure for implementing the DWA local planner algorithm can be found in ROS Wiki [38].

조선대학교
CHOSUN UNIVERSITY

4. SIMULATION AND HARDWARE SETUPS

In this chapter, the operation of mapping, localization, and navigation of a mobile robot using ROS shall be covered. Before delving into the simulated and experimental implementations, the organization of the simulation and experiment setups, as well as the structure of ROS navigation, will be discussed. ROS integration with the mobile robot and Velodyne lidar sensor, and also ROS packages for SLAM and autonomous navigation, will also be addressed.

4.1. Simulation Setup

The simulations are done with Gazebo Simulator to visualize the 3D real robot movement, a physical engine, and Rviz to visualize the robot's internal state and sensor data.

4.1.1. *Model of Robot*

The mobile robot used for simulation is known as "MRP NRLAB02," and it includes packages for controlling the robot during simulation and experimentation. In this thesis, a package is used to observe the completeness of the navigation of mobile robots in the simulation environment using the robot’s URDF representation which is shown Figure 4-1.

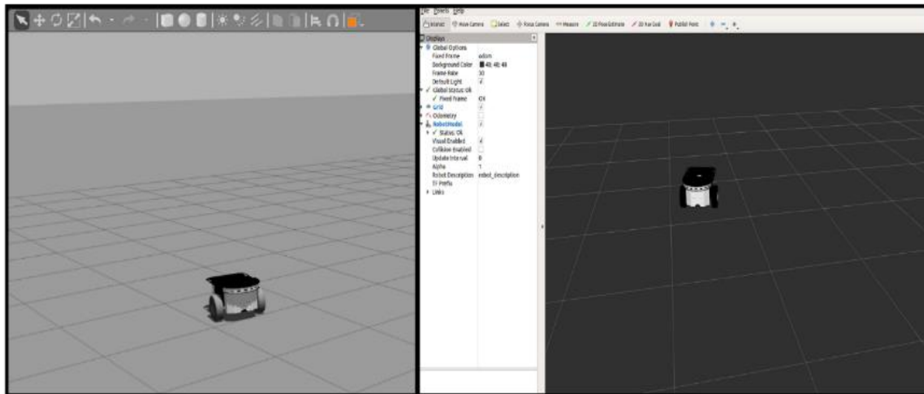


Figure 4-1 The simulated platform of the robot on Gazebo and Rviz from left to right.

4.1.2. *Interfacing of Simulated Robot and Lidar Sensor*

In this thesis, the Unified Robot Description Format (URDF) of the Hokuyo 2D lidar sensor is used to interface it with the mobile robot's URDF (Figure 4.2) for simulation. The

조선대학교
CHOSUN UNIVERSITY

URDF of MRP_NRLAB02 is modified by adding the laser scanner plugin model link, joint, and inertia to visualize the laser information. The URDF file is written in such a way that the robot's each link is the parent link's child (Motor baseplate in our case), with each link connected by joints. The offset from the parent link's reference frame and the axis of rotation is used to define joints, as shown in Figure 4-3 and Figure 4-4.

The simulated Hokuyo lidar sensor has 180^0 horizontal field view and a range of detection up to 15 meters. The rays used have 720 sample points, and the topic published from this lidar is */hokuyo_lidar/scan*, with a message type of *sensor_msgs/LaserScan*.

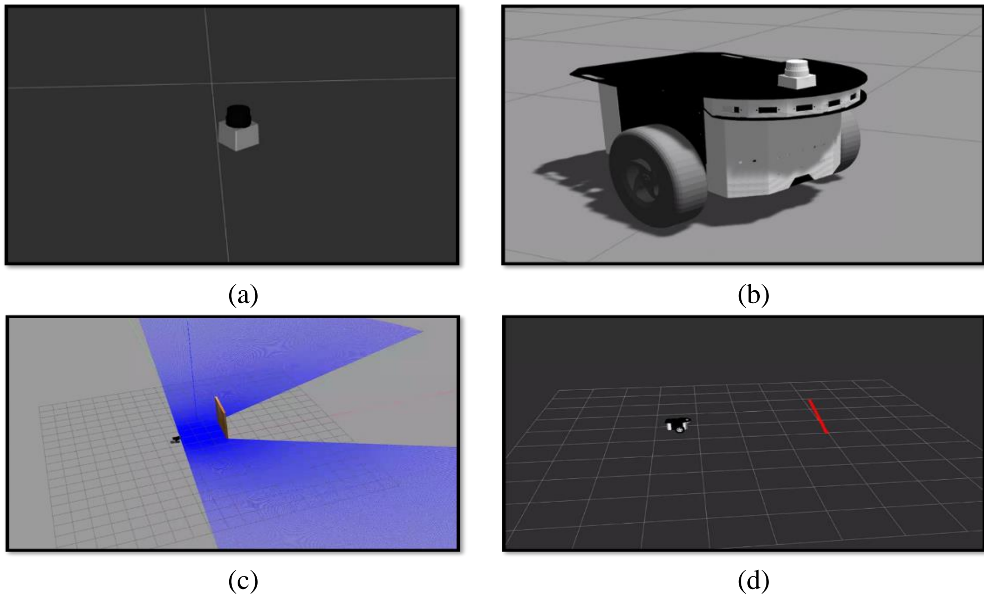


Figure 4-2 The URDF of Hokuyo lidar sensor and mobile robot in a simulated environment (a) URDF of Hokuyo lidar sensor on Rviz. (b) Mounting the lidar sensor on the top of the robot using URDF on the Gazebo simulator. (c) Facing the wall (the light brown color) as an obstacle in front of the robot to observe the lidar output. The blue color is the ray of the lidar activating Gazebo's ray visualization plugin.(d) Visualizing the internal state of the laser scan (the red color shows the detected obstacle in front of the robot from (c)) on Rviz.

조선대학교
CHOSUN UNIVERSITY

```xml
<!--Hokuyo Laser link-->
<link name="hokuyo_link">
<inertial>
  <origin xyz="0 0 0" rpy="0 0 0" />
  <mass value="0.01" />
  <!-- RANDOM INERTIA BELOW -->
  <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0" izz="1e-6"/>
</inertial>

<visual>
  <origin xyz="0 0 0" rpy="0 0 0" />
  <geometry>
    <mesh   filename="package://mrp_nrlab02/meshes/hukuyo.dae" />
  </geometry>
  <material name="white" />
</visual>

<collision>
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <geometry>
     <box size="0.1 0.1 0.1"/>
  </geometry>
</collision>
</link>

 <!--Hokuyo Laser joint-->
<joint name="hokuyo_joint" type="fixed">
  <origin xyz="0.167 0.013 0.25"  rpy="0 0 0" />
  <parent link="Motor_baseplate" />
  <child  link="hokuyo_link" />
  <axis   xyz="0 0 0" />
</joint>
```

Figure 4-3 URDF representation of Hokuyo lidar sensor.

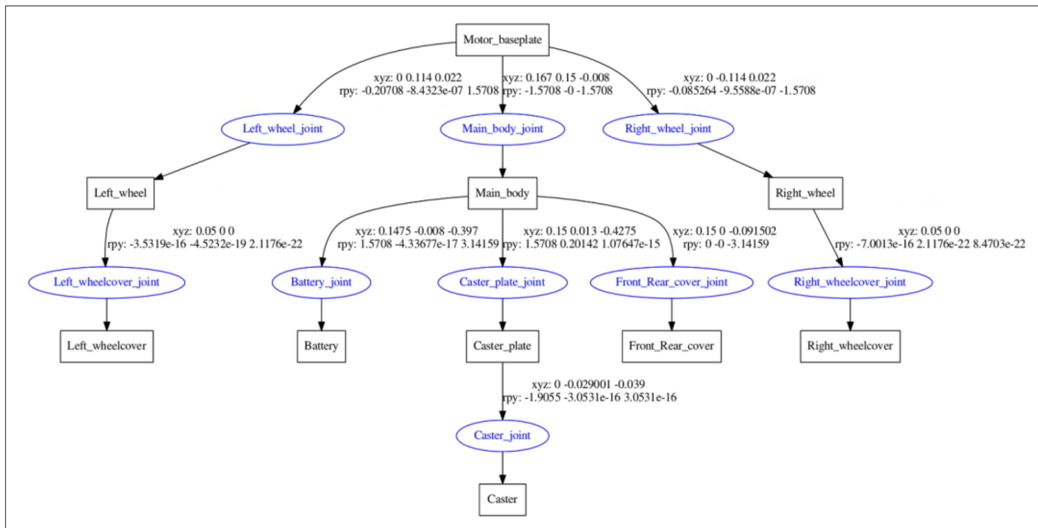
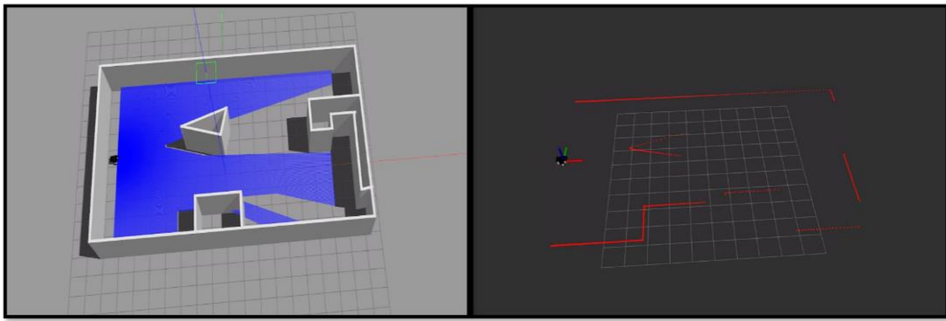


Figure 4-4 Transformation tree diagram of the mobile robotSimulation Environment used for Navigation of Mobile Robot.

4.1.3. ***Simulation Environment used for Navigation of Mobile Robot***

Before testing in the real environment is done, the environment for simulation is constructed on Gazebo. The simulation environment can be seen in Figure 4-5, with $17.5m\ x\ 10.1m$ (approximately an area of $176.75m^2$). The simulation environment will be used later in the result section for mapping, localization, and path planning.

조선대학교
CHOSUN UNIVERSITY



(a) (b)

Figure 4-5 Simulation Environment. (a) The built mobile robot inside the environment and lidar rays using Gazebo. (b) Rviz's internal lidar signal visualization by detecting obstacles in the environment.

4.2. Hardware Setup

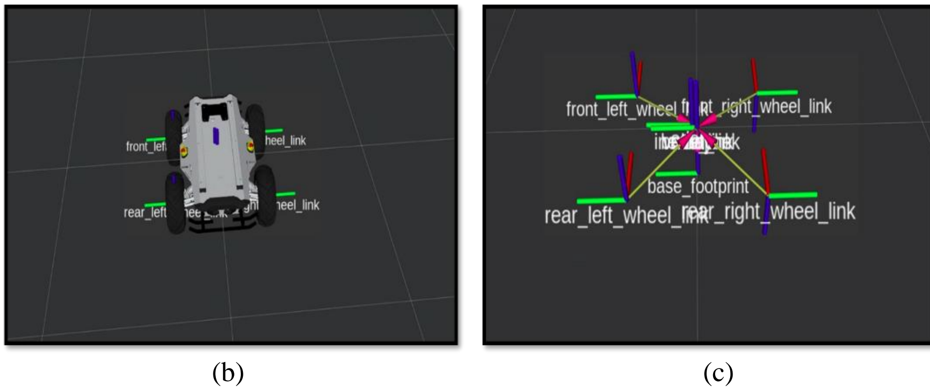
Instead of using the Gazebo simulator, the Rviz simulator is utilized to display the internal state of the robot, and real-world situations are used to implement the algorithms for the navigation of the mobile robot autonomously.

4.2.1. *Robot Platform*

The mobile robot used for the experiment is a four-wheeled differential robot called “Scout Mini”. The robot platform is shown in Figure 4.6 and the hardware specification is listed in Table 4-1.

(a)

조선대학교
CHOSUN UNIVERSITY



(b) (c)

Figure 4-6 The hardware platform of the mobile robot used for experimentation. (a) Robot Hardware setup, (b) Mobile robot on Rviz, and (c) Frame transformation on Rviz.

Table 4-1 Hardware specification.

Dimension	627 X 550 X 252 mm
mode of operation	4-wheel Differential-Drive Model
Wheelbase	452mm
Minimum ground clearance	107mm
Battery	24V / 15Ah
Maximum speed	10km/h
Communication environment	Standard CAN / RS232

Figure 4-6 (a) depicts the Scout Mini mobile robot with the USB-HUB expansion interface, USB to CAN module, Intel RealSense D435, LCD screen, and Velodyne VLP-16 installed at the top layer. Through the aviation expansion interface of the chassis, Scout Mini provides a power and communication interface for the upper equipment. The Scout Mini chassis power expansion interface is powered by the chassis' battery, which has a maximum power output support of 24V and 5A. It has voltage regulator modules of 19V and 12V. The 19V voltage regulator module primarily powers the Nvidia AGX Xavier, while the 12V voltage regulator module powers the VLP-16, USB-HUB, and wireless routers. Figure 4.7 below shows the external connection topology of Scout Mini.

조선대학교
CHOSUN UNIVERSITY

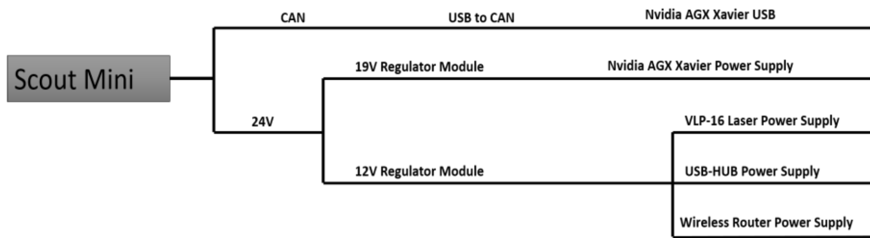


Figure 4-7 External topology connection of Scout Mini.

The external topology connection of Nvidia AGX Xavier as the core computing unit is shown in Figure 4.8. Xavier's network port is connected to the router's network port, which is convenient for remote desktop connection, access, and debugging and easy to expand to other network devices. The USB-HUB mainly expands and connects D435 binocular camera, and LCD camera.

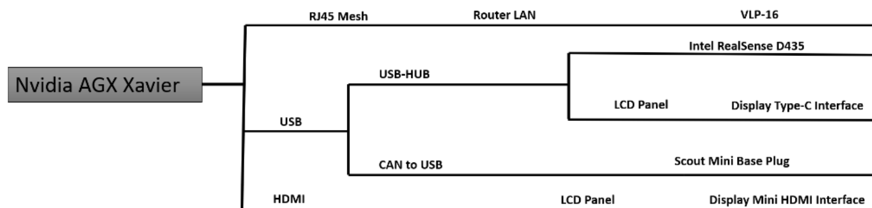


Figure 4-8 Nvidia AGX Xavier external topology connection.

4.2.2. *Sensors on Scout Mini*

i. **Intel ReasSense D435 Camera:** Binocular vision sensors are used in diverse robot applications, including robot vision measurement and visual navigation. The Intel RealSense Depth Camera D435 effectively captures and broadcasts depth data from objects that are moving and allows mobile prototypes to have very accurate depth perception. It has a broad field of view and a global image shutter. The hardware specifications of the D435 camera are detailed in Table 4-2 and Figure 4-9 depicts the visual image of D345.

조선대학교
CHOSUN UNIVERSITY

Table 4-2 Hardware specification of Intel RealSense D435 camera.

	Model	Intel Realsense D435
Basic Features	Application Scenario	Outdoor/Indoor
	Measuring distance	About 10m
	Depth shutter type	Global shutter (3um x 3um)
	Whether to support IMU	Support
Depth camera	Deep tech	Affinity infrared
	FOV	86^0 x 57^0 (±3)
	Minimum depth distance	0.105m
	Depth detection	1280 x 720
	Maximum measuring distance	About 10m
	Depth frame rate	90 fps
RGB	Resolution	1280 x 800
	FOV	69.4^0 x 42.5^0 (±3^0)
	Frame rate	30 fbs
Other information	Size	90mm x 25mm x 25mm
	Interface type	USB-C 3.1

Figure 4-9 Intel RealSense D435 camera

ii. **Velodyne VLP-16 Laser Sensor:** The VLP-16 is Velodyne's most advanced and smallest lidar. The VLP-16 is less expensive than comparable sensors while retaining some of the basic features of Velodyne's breakthrough lidar, such as real-time, 360° field of view, 3D coordinates and distance, and reflectance measurements with calibration. The VLP-16 has a measuring range of up to 100 m, a low power consumption (8 W), is lightweight (830 g), has a small size (103mm x 72mm), and has dual return capability, making it ideal for man-machine mounts and other mobile devices. The hardware specifications of the

조선대학교
CHOSUN UNIVERSITY

VLP-16 are detailed in Table 4-3 and Figure 4-10 depicts the visual image of the VLP-16.

Table 4-3 Hardware specification of VLP-16.

VLP-16	Properties
Maximum range	100m
Range accuracy	±3cm
Scan rate	300,000 points/sec
Vertical viewing angle	$-15^0 \sim +15^0$
Number of channels	16
Scanning frequency	5Hz – 20Hz
Horizontal field of view	360 degrees
Weight	830g
Power consumption	8W
Voltage	9V – 18V
Operating temperature	$-10^0C \sim +60^0C$

Figure 4-10 Velodyne VLP-16 lidar sensor.

4.3. Structure of ROS Navigation

ROS is an open-source platform that could be considered a middleware (Figure 4-11) that provides high-level abstraction between low-level hardware and drivers and high-level software APIs.

조선대학교
CHOSUN UNIVERSITY

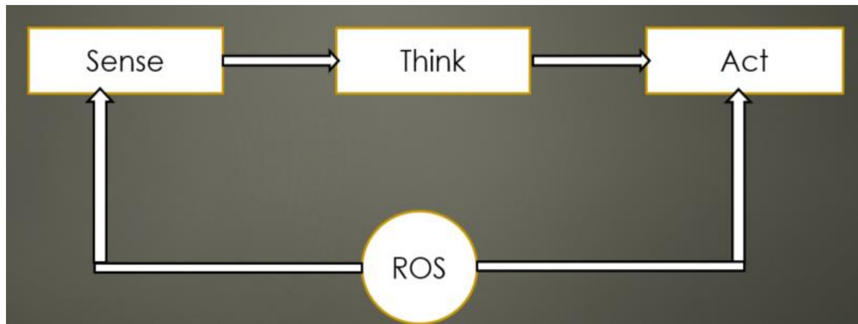


Figure 4-11 Robot process cycle using ROS

The problem of navigation is fundamental in robotics and other important technologies. For the sake of making the mobile robot navigate autonomously, it must first understand where it is, where it is going, and how it will get there. The major challenges that the mobile robot faces while navigating in an environment are depicted in Figure 4-12, and these challenges will be discussed in the following sections with ROS.

ROS navigation stack is used to accomplish mobile robot navigation from one point to a target point. The ROS navigation stack is a collection of ROS nodes and algorithms that are used to autonomously move a robot from one location to another while avoiding any obstacles in its path. In the coming subsection, each algorithm used in ROS will be explained including SLAM_gmapping, Adaptive Monte Localization, and Path Planning.

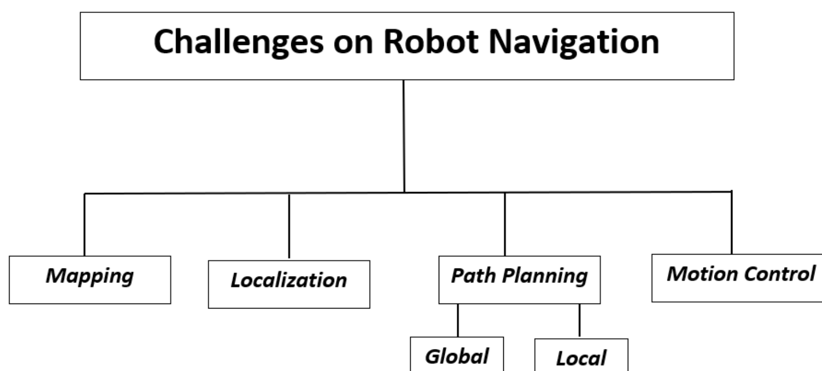


Figure 4-12 Challenges to Navigation of mobile robot

조선대학교
CHOSUN UNIVERSITY

The followings are the input for the navigation stack to send commands to our robot to move to its desired goal:

- **Current pose:** The current orientation and position of the robot.
- **Goal pose:** The robot's desired location for achieving its goal.
- **Odometry Data:** Using the motion sensor's data for estimating the change in pose over time. These sensors include wheel encoders, IMU, GPS, etc…
- **Laser Sensor:** Data from the laser (lidar) sensor is needed to recognize objects in the environment.

Taking the above as an input, the navigation stack in exchange will output the velocity commands that are needed and send them to the mobile base for moving the robot to the designated target position. In summary, the basic goal of the navigation stack is to move the robot from its starting location to its destination while avoiding collisions with objects and getting lost on its way. The diagram (Figure 4-13) shows the basic building blocks of the Navigation stack taken from the ROS official website. The sections below provide a quick overview of all the blocks that have to be submitted as input to the ROS Navigation stack.

i. Odometry Source

Odometry information refers to an estimated pose of the robot and velocity in free space. The determination of the odometry information is done through kinematics from the robot's motor shafts encoder counts. The robot's odometry data determines its position in relation to its starting position. The primary sources of odometry are IMUs, wheel encoders, 2D/3D cameras (for visual odometry), and GPS. We used the information from a wheel encoder to publish the *odom* value (Figure 4-14) to the navigation stack with x *nav_msgs/Odometry* message type that can hold the robot's position and velocity. The *odom* data will then be used by the ***SLAM_gmapping*** algorithm to create the 2-dimensional map of the environment.

조선대학교
CHOSUN UNIVERSITY

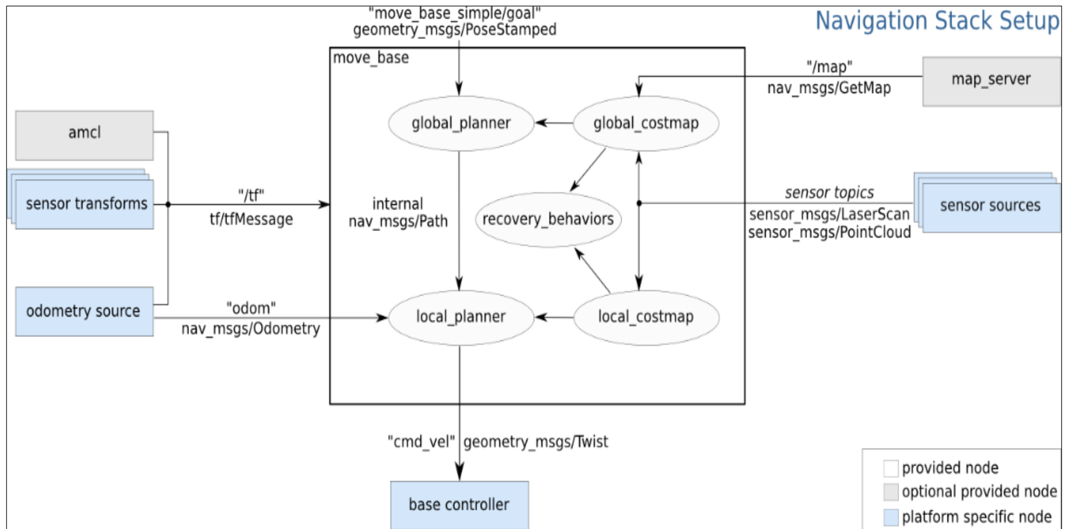
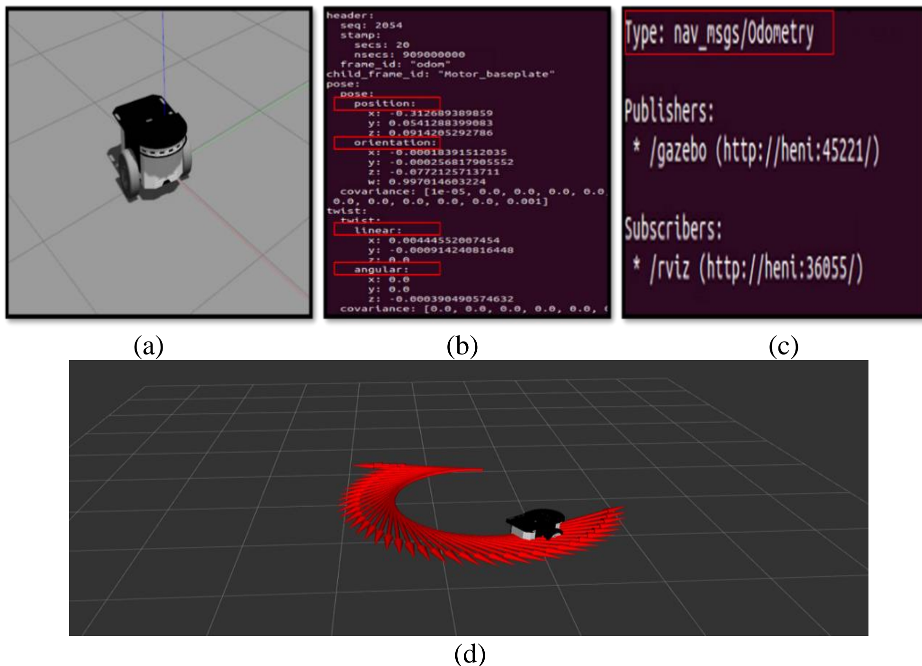


Figure 4-13 ROS Navigation stack structure



(a) (b) (c)

(d)

Figure 4-14 Odometry source information of a mobile robot. (a) Shows the two-wheeled mobile robot that is used for simulation in this thesis (MRP-NRLAB02 Red-One technology). (b) The odometry data from the mobile robot's wheel encoder containing position, orientation, linear velocity, and angular velocity with topic name */odom*, and the data can be visualized with ROS command of *$ rostopic echo /odom*. (c) It shows the type of message that the */odom* topic contains with nav*_msgs/Odometry's* message type and it can be extracted with the ROS command of *$ rostopic info /odom*.(d) Shows the visual representation of odometry information.

조선대학교
CHOSUN UNIVERSITY

ii. Sensor Sources

Data provided from the sensors is used by the navigation stack to perform two tasks: the first is to locate the robot on a map using a 2-dimensional laser sensor with a message type of *sensor_msgs/LaserScan* or a 3-dimensional laser sensor with a published message *msgs/PointCloud* of the sensor, and the second is to detect and avoid obstacles in the robot's path through the surrounding environment. In this thesis, the Hokuyo 2D lidar sensor's URDF is used to interface it with the mobile robot's URDF (Figure 4-16) for simulation, and the Velodyne lidar Puck (VLP-16) lidar sensor is used for practical implementation with a 4-wheeled differential mobile robot called Scout Mini.

iii. Sensor Transformation (Transform Configuration)

Sensor transformation is critical when working with mobile robots because the robot must be aware of both itself and its surroundings. To accomplish this, the robot must be able to calculate its orientation and position relative to the obstacle relative to the obstacle. The process of specifying how data expressed in one frame can be transformed into a different frame is known as coordinate transformation. For instance, provided that the laser detects an obstacle with the lidar sensor at 20cm in the front (illustrated in Figure 4-15), this means that it is 20cm from the laser, but not from the center of the robot (base_link of the robot). We must convert the 20cm from the lidar frame to the robot frame depicted by the dark yellow arrow to determine the distance from the robot's center.

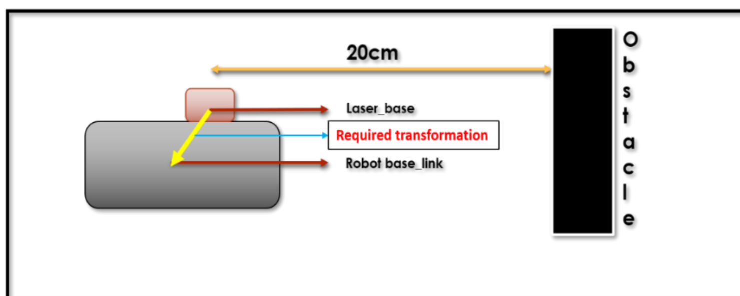


Figure 4-15 The transformation between the coordinates of laser and robot base link

조선대학교
CHOSUN UNIVERSITY

For comparing data from different sensors, the data collected by numerous robot sensors should be referenced to a common reference frame(in our case, the base link frame id name of the robot used for simulation is *Motor_baseplate* (Figure 4-16)). Using ROS transforms, the relationship that is between the main robot coordinate frame (Motor_baseplate) should be published by the robot and the various sensor frames (for the simulated robot case hokuyo_link).

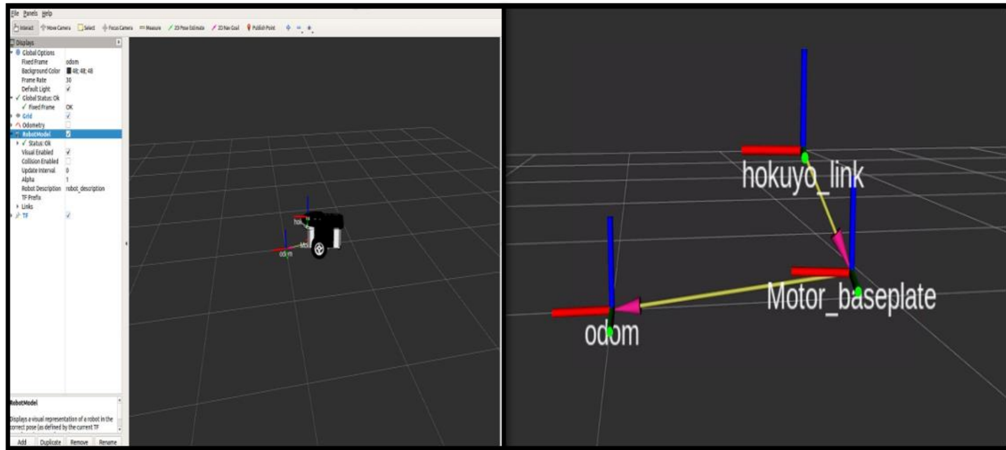


Figure 4-16 URDF of lidar sensor transformation with the simulated robot on Rviz

The yellow arrow in Figure 4.8 represents the transform visualization from the lidar base link (hokuyo link) to the mobile robot's base link frame (Motor baseplate), which is then transformed to the robot's odometry (odom frame).

i. Base Controller

The navigation stack requires a trajectory planner to send velocity commands to the base of the robot coordinate frame using a *geometry_msgs/Twist* message with the "*cmd_vel*" topic (for the simulated robot, the topic name is */nrlab/cmd_vel*). This necessitates the existence of a node that subscribes to this topic and is capable of taking linear velocities along the x and y axes and converting them into motor commands for a mobile robot. The primary purpose of the base controller is to convert the navigation stack's output, that is the message called Twist to the respective motor velocities of the robot.

조선대학교
CHOSUN UNIVERSITY

Each ROS algorithm, including SLAM_gmapping, Adaptive Monte Localization (AMCL), and Path Planning, will be explained in the following subsections.

4.3.1. ***SLAM-gmapping***

SLAM is the process of creating the environment's map while keeping track of the position of the robot on the map. This problem is basically what mapping is solving. For reducing the common depletion problem that is associated with the Rao-Blackwellized particle filter, an adaptive resampling technique is employed by the gmapping package [32, 40, 41]. To create a map, a two-dimensional occupancy grid method is used by the gmapping package. An obstacle is inserted into a cell or the cell is cleared using sensor stream data. Clearing a cell for each successful laser-scan sample includes ray-tracing through a grid. By comparing current laser scans to prior laser scans, GMapping can help decrease and fix odometry drift faults.

The gmapping ROS package is an implementation of the SLAM algorithm. It is used to generate a 2D map from the robot's lidar sensor and odometry data as it moves around the area. It also includes a slam gmapping node, which reads data from the laser and transforms it from the laser source to the base link, broadcasting the transform from the "map" to "odom" frames, resulting in an occupancy grid map (OGM). To obtain the data needed to build a map, the slam_gmapping node subscribes to the laser topic (*/hokuyo_lidar/scan* topic for simulation and */scan* topic for hardware implementation) and an extensive transform topic (*/tf*). Throughout the process of slam_gmapping, the generated map is published into the /map topic, which uses the message type *nav_msgs/OccupancyGrid*. An occupancy grid map is shown in Figure 4-17. Occupancy is expressed as an integer in the range [0, 100], with 0 (entirely free or white color), 100 (entirely occupied or black color), and -1 for an unknown location.

조선대학교
CHOSUN UNIVERSITY

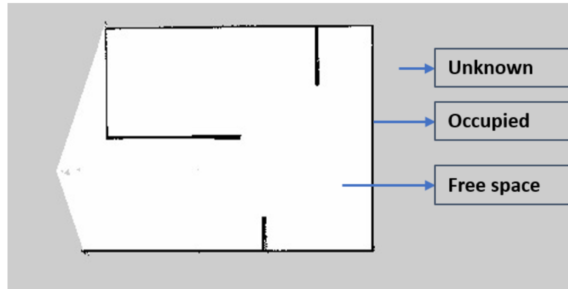


Figure 4-17 Example of Occupancy Grid Map (OGM).

The map_server package, which is part of the ROS navigation stack, saves the OGM map output and provides the *map_saver* node that allows the access to map data from the ROS service, and saves it into a file. As shown in Figure 4-18, the map data is saved in two files:

a. YAML file: containing the image name and map metadata.
b. PGM image: the image itself with the encoded data of the OGM.

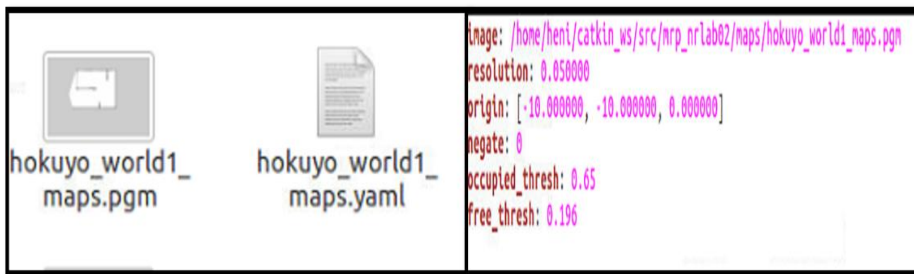


Figure 4-18 The PGM file and the metadata of the YAML file.

The command ***rosrun map_server map_saver -f name_of_map*** can be used to save the map and the resulting map is a static map that is required for localization and path planning. The gmapping ROS package includes a set of parameters for modifying the SLAM algorithm's behavior described in Chapter 3 [42]. The map's size does limit the selectable area for exploration in gmapping; inorder for exploring the whole environment at once, the size of the map must be regulated accordingly. The proper configuration of the parameters is essential to building a proper map. Without a proper configuration of the robot with the algorithm, it will be impossible for us to create a good map of the environment, and without a good environment, it will not be possible to properly navigate the robot inside the environment.

조선대학교
CHOSUN UNIVERSITY

To build the map with gmapping, we must meet two requirements: the first is to provide good laser data (Velodyne lidar is used for real-time implementation), and the second is to provide good odometry data (in our case wheel encoder is used for odometry data).

Table 4-4 Major parameters of gmapping

General Parameters	
base_frame (string, default: “base_link”)	• It shows the name of the frame attached to the base of the mobile robot.
map_frame (string, default:”map”)	• It is the name of the frame attached to the map.
odom_frame (string, default:”odom”)	• It indicates the name of the frame attached to the odometry system (wheel encoder in our case).
map_update_interval (float, default: 5.0)	• It sets the time (in second) to wait until update the map. • Lowering this number updates the occupancy grid more often with greater computational load.

Laser Sensor Parameters	
maxRange (float)	• It sets the maximum range of the laser.
maxUrange (float, default: 80.0)	• It is used to set the maximum usable range of the laser. • The laser beam will be cropped to this value range.
minimumScore (float, default: 0.0)	• It is used to set the minimum score to consider a laser reading good. • For a laser scanner with a limited range, it can also avoid jumping pose estimates in large open spaces.

Map Dimensions and Resolutions	
xmin (float, default: -100.0)	• It is the minimum x-axis map size in meters.
ymin (float, default: -100.0)	• It is the minimum y-axis map size in meters.
xmax (float, default: 100.0)	• It is the maximum x-axis map size in meters.
ymax (float, default: 100.0)	• It is the maximum y-axis map size in meters.
delta (float, default: 0.05)	• It sets the resolution of the map in meters per occupancy grid block.

Other Additional Parameters	
linearUpdate (float, default: 1.0)	• It is used to specify how far the robot must move translate in order to process a laser reading.
angularUpdate (float, default: 0.5)	• It specifies the angular distance at which the robot must rotate in order to process a laser scan reading.
temporalUpdate (float, default: -1.0)	• It is used to set the time to wait between a scan readings. • If it is set to –1.0, its function is disabled.
particles (int, default: 30)	• It is the number of particles in the filter.

조선대학교
CHOSUN UNIVERSITY

Aside from that, the coordinate transformation between the lidar sensor's base and the robot's base is required, as shown before in Figure 4-16. Table 4-4 shows the major parameters that must be configured while mapping and the effect of the parameters on map quality will be discussed in the results section.

4.3.2. *Adaptive Monte Carlo Localization*

Determining the robot’s pose inside a mapped environment is known as localization. Robot Localization occurs when a robot moves around a map and needs to know its position and orientation within the map using sensor readings. The Monte Carlo Localization (MCL) from the previous chapter is the most popular algorithm in robotics, after the deployment of MCL, the robot would be navigating through its known map and collect sensory information by the use of range-finder sensors and RGB cameras. Then, the MCL will use this sensor for measuring and keeping track of the pose of the robot [43].

MCL is also known as Particle Filter Localization because it uses particles for locating the robot. Because the robot does not always move expectedly, it creates a huge number of random estimates about where it will go, which is the next pose. As shown in Figure 4.19, these guesses are known as particles; every particle has an orientation and a position and represents a guess as to where the robot could be located. The particles depicted by the red arrow in Figure 4-19 are used to estimate the pose of the robot.

The ROS package AMCL [44] includes the amcl node, which employs the MCL algorithm to track the location of the movement of the robot in a two-dimensional space. This node subscribes to laser data, the laser-based map, and the robot's transformation, and then publishes the estimated pose on the map.

조선대학교
CHOSUN UNIVERSITY

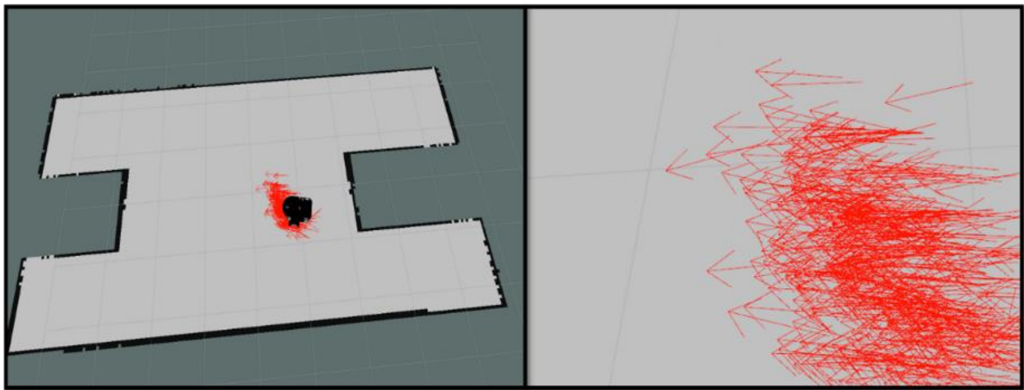


Figure 4-19 Examples of AMCL

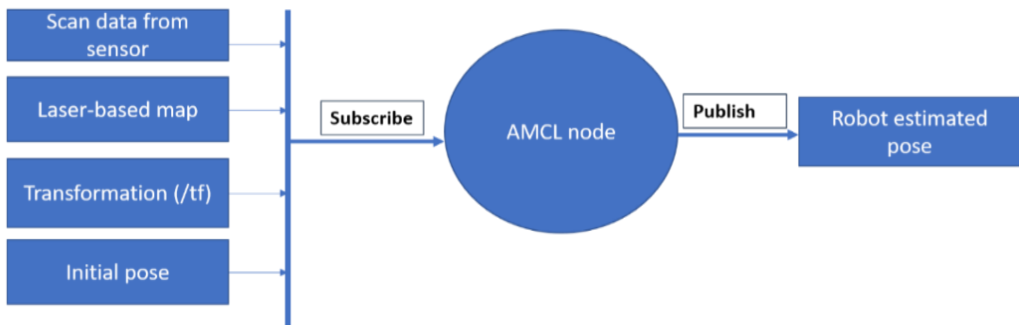


Figure 4-20 AMCL node data subscription and publication

As shown in Figure 4-20, three basic requirements must be met to properly localize a mobile robot within a map: providing good laser data with the proper transformation of its base link with the robot base link, good odometry data from the encoder or IMU, and properly constructed laser-based map data. Furthermore, as with the mapping algorithm, the proper parameter configuration is critical for localizing a mobile robot in its environment. The amcl node, most importantly, has two major requirements for the robot's transformation, which are as follows:

조선대학교
CHOSUN UNIVERSITY

i. The amcl converts an incoming laser scan with the laser's frame id to an odometry frame, which needs a path from the frame where the laser scans are published to the odometry frame through the tf tree.

ii. AMCL searches the transformation between the laser frame and the base frame and latches it forever to ensure proper localization. This means that a moving laser relative to the base can't be handled by the amcl node and requires a fixed or static transformation between the two frames.

The major parameters of the amcl algorithm that are useful for proper localization of the robot in the environment are shown in Table 4-5, and the effect of the parameters on localization quality will be discussed in the results section.

Table 4-5 Major parameters of AMCL.

General Parameters	
base_frame_id (string, default: "base_link")	• It shows the name of the frame attached to the base of the mobile robot.
global_frame_id (string, default:"map")	• It indicates the name of the coordinate frame published by the localization system.
odom_frame (string, default:"odom")	• It indicates the name of the frame attached to the odometry system (wheel encoder in our case).
use_map_topic (bool, default: false)	• It shows if the node gets the map data from topic or from a service call. • When this property is set to true, the amcl node will subscribe to the map topic instead of making a service call.

조선대학교 CHOSUN UNIVERSITY

Odometry Model Parameters	
odom_model_type (string, default: "diff")	• It sets the odometry model to use. This model could be "diff", "omni", "diff-corrected" or "omni-corrected"
odom_alpha1 (double, default: 0.2)	• It is used to specify the expected noise in the odometery's rotation estimate from the rotational component of the robot's motion.
odom_alpha2 (double, default: 0.2)	• It sets the expected noise in the odometry's rotation estimate from the translational component of the robot's motion.
odom_alpha3 (double, default: 0.2)	• It reflects the expected noise in the translation estimate of the odometry from the translational component of the robot's motion.
odom_alpha4 (double, default: 0.2)	• It reflects the expected noise in the translation estimate of the odometry from the rotational component of the robot's motion.
odom_alpha5 (double, default: 0.2)	• It is translation-related noise parameter (only used for omni robot model)
laser_z_rand (double, default: 0.05)	• It is the mixture weight for the z_rand part of the model.
laser_likelihood_max_dist (double, default: 2.0)	• It is the maximum distance in meters to do obstacle inflation on map, for use in likelihood_field model.

Filtering Parameters	
min_particles (int, default: 100)	• It is the minimum number of particles allowed for the filter
max_particles (int, default: 5000)	• It is the maximum number of particles allowed for the filter
kld_err (double, default: 0.01)	• It is used to specify the maximum error allowed between the true distribution and the estimated distribution.
resample_interval (int, default: 2)	• It is used to set the number of filter updates required before resampling.
update_min_d (double, default: 0.2)	• It sets the linear distance in meters that the robot has to move in order to perform a filter update.
update_min_a (double, default: $\frac{\pi}{6.0}$ radians)	• It is the rotational movement or angular distance in radians that the robot has to move to perform a filter update.
gui_publish_rate (double, default: -1.0)	• It is maximum rate in Hz at which scans and paths are published for visualization. • -1.0Hz means it is disabled.
transform_tolerance (double, default: 0.1seconds)	• It reflects time in seconds, which is used to post-date the published transform, indicating that it is valid in the future.

조선대학교
CHOSUN UNIVERSITY

4.3.3. ***Move-Base Package***

Move base is a key component of the ROS navigation stack that connects all planner and controller behaviors. The move base package includes the *move_base* node, which moves the robot from its current position to the target point. It is a *SimpleActionServer* implementation that accepts a target pose with *geometry_msgs/PoseStamped* message type so that *SimpleActionClient* can send a target point to this node. One of the topics provided by the move base action server is *move_base/goal*, which is the navigation stack's input that will be used to provide the goal pose. Table 4-6 shows some of the moving base [45] parameters that are responsible for the planner and controller frequencies.

Table 4-6 Move base parameters

controller_frequency (double, default: 20.0)	• **It is the frequency in Hz at which the control loop runs and velocity commands are sent to the base.**
controller_patience (double, default: 15.0)	• **It indicates how long the controller will wait in seconds without receiving a valid control before performing space-clearing operations.**
planner_frequency (double, default: 0.0)	• **It specifies the frequency in Hz at which the global planning loop should be run.** • **If set to 0.0, the global planner will only run when a new goal is received or when the local planner reports that its path is blocked.**
planner_patience (double, default: 5.0)	• **It indicates how long the planner waits in seconds in an attempt to find a valid plan before performing space-clearing operations.**

The frequency is determined by the computer's computation power and the quality of the communication. If the values of these parameters are not properly configured, they may cause a jerk motion. This jerking behavior is caused primarily by a lack of time to complete the computation, as well as the fact that when the controller completes its task, it must wait for the next one. When the robot becomes stuck, the controller and the waiting time of the controller or planner before the computation is restarted if determined by the planner's patience.

조선대학교
CHOSUN UNIVERSITY

As stated in the previous description, sending a goal to the move base node activates some other processes that involve some other nodes that result in moving the robot to the target pose. For the sake of using the move_base node in the navigation stack, a local and global planner is needed.

4.3.3.1 Global Planner on ROS

When the move base node receives a new goal, it immediately sends it to the global planner. The global planner is responsible for the safe path calculation depicted in Figure 4-21 to achieve the desired goal pose. This path is calculated before the movement of the robot starts; it doesn't consider the readings made by the robot's sensors while it is moving. The light green line in Figure 4-21 represents the global path to take to reach the goal position.

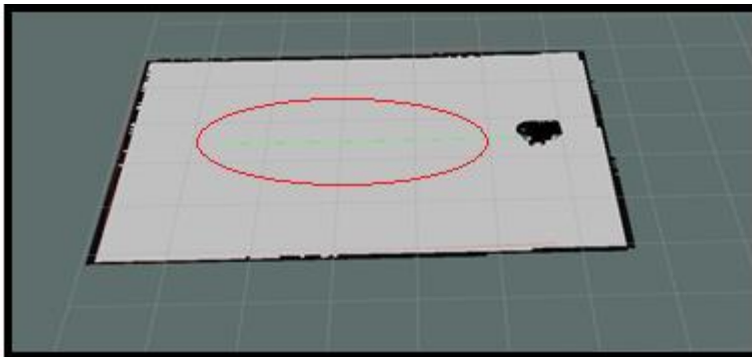


Figure 4-21 Example of a global planner

Depending on the setup the robot use and the environment it navigates, there exist three global planners that adhere to the nav_core:: BaseGlobalPlanner interface: Navfn, global_planner, and carrot_planner [46].

- **Navfn:** it is the most widely used global planner for navigation in ROS. It employs Dijkstra's algorithm is employed, as described in section 3.1.5.1, to determine the shortest path with the lowest cost between the initial and goal poses.

조선대학교
CHOSUN UNIVERSITY

- **CarrotPlanner: :** it is the simplest, take the target pose and check whether it's an obstacle, then choose an alternate goal that's close to the original by going back along the vector between the target point and the robot. This planner is useful if we need our robot to move near a given target, even though the target is out of reach. This planner is ineffective in a complex indoor environment.
- **GlobalPlanner** It is a more versatile and customizable replacement for navfn. And also, it includes more options such as toggling grid path, toggling quadratic approximation, and support for A*. Navfn and global planner are based on [47], which allows us to change the algorithm that navfn uses to calculate the path.

Table 4-7 Global planner parameters

Global_Planner Parameters	
use_dijkstra (bool, default: true)	• If true, use dijkstra's algorithm. Otherwise, A*
use_quadratic (bool, default: true)	• If true, use the quadratic approximation of the potential. Otherwise, use a simpler calculation.
use_grid_path (bool, default: false)	• If true, create a path that follows the grid boundaries. Otherwise, use a gradient descent meth
old_navfn_behavior (bool, default: false)	• If we want global_planner to exactly mirror the behavior of navfn, set this to true (and use the defaults for the other boolean parameters)
lethal_cost (int, default: 253)	• Lethal Cost (dynamic reconfigure)
neutral_cost (int, default: 50)	• Neutral Cost (dynamic reconfigure)
cost_factor (double, default: 3)	• Factor to multiply each cost from costmap by (dynamic reconfigure)

Table 4-7 shows some of the major navfn [48] and global_planner [49] parameters.

Navfn Parameters	
allow_unknown (bool, default: true)	• Specifies whether or not navfn should be allowed to create plans that traverse unknown space.
planner_window_x (double, default: 0.0)	• Specifies the x size of an optional window to restrict the planner to. • This can be useful for restricting NavFn to work in a small window of a large costmap.
planner_window_y (double, default: 0.0)	• Specifies the y size of an optional window to restrict the planner to. • This can also be useful for restricting NavFn to work in a small window of a large costmap.
default_tolerance (double, default: 0.0)	• A tolerance on the goal point for the planner. • NavFn will attempt to create a plan that is as close to the specified goal as possible but no further than default_tolerance.
visualize_potential (bool, default: false)	• Specifies whether or not to visualize the potential area computed by navfn via a PointCloud2.

조선대학교
CHOSUN UNIVERSITY

When the planner creates a trajectory, it must be done in accordance with a map. The global planner calculates its path using a map called a costmap. A costmap is a map that represents places in a grid of cells using binary values that represents either a free space or the presence of an obstacle. The costmap values are binary, representing either free space or paces where the robot would collide. In ROS, costmap consists of an *obstacle map layer, static map layer*, *and inflation layer*.

- **Static map layer**: It is the fixed map given as an input to the navigation stack.
- **Obstacle map layer**: It includes 2-dimensional and 3-dimensional obstacles (voxel layer).
- **Inflation layer:** It is where the cost of each 2D costmap cell is calculated by inflating obstacles.

There exist two types of costmap: local costmap and global costmap. The global costmap depicted in Figure 4-22 will be discussed in this section, while the local costmap will be presented in the following section.

Global Costmap

A global costmap is constructed from the map obtained by SLAM gmapping by inflating the obstacles on the map provided to the navigation stack. The costmap is initialized to match the static map's height, width, and obstacle data. The global costmap has its own set of parameters (Some of these are listed in Table 4-8 that should be optimized as much as possible. The global planner uses the global costmap to calculate the path to follow.

조선대학교
CHOSUN UNIVERSITY

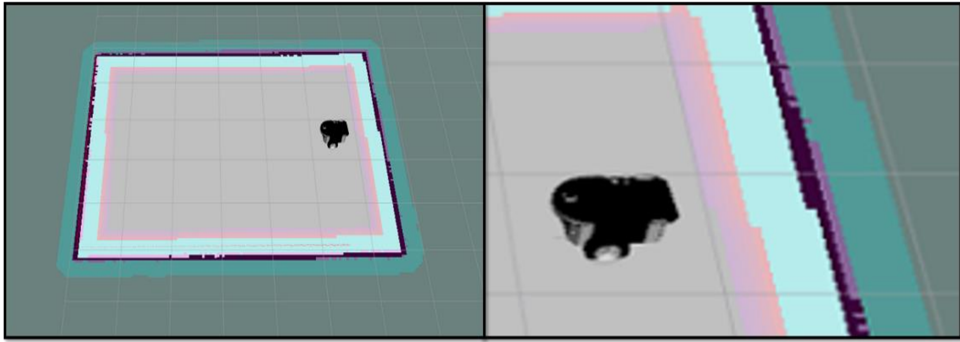


Figure 4-22 Example of global costmap

Table 4-8 Global costmap parameters.

Global Costmap parameters	
global_frame (string, default: "/map")	• The global frame for the costmap to operate.
robot_base_frame (string, default: "base_link")	• The name of the frame used for the base link of the robot.
rolling_window (bool, default: false)	• Whether or not to use a rolling window version of the costmap.
max_obstacle_height (double, default: 2.0)	• the maximum height of any obstacle to be inserted into the costmap in meters.
obstacle_range (double, default: 2.5)	• The default maximum distance from the robot at which an obstacle will be inserted into the cost map in meters.
raytrace_range (double, default: 3.0)	• The default range in meters at which to raytrace out obstacles from the map using sensor data.
cost_scaling_factor (double, default: 10.0)	• A scaling factor to apply to cost values during inflation.

4.3.3.2 Local Planner on ROS

Once the global planner has calculated the path to take, this path is given to the local planner, which then executes each segment of the global plan. This means that given a plan to follow provided by the global planner and a map, the local planner will provide a velocity command to move the robot. The local planner (shown in Figure 4-23 with a yellow line), as opposed to the global planner, monitors odometry and lidar data and selects a collision-free local plan for the robot by recomputing the path to follow to keep the robot from colliding with objects.

조선대학교
CHOSUN UNIVERSITY

In addition to the global planner, there are various types of local planners depending on the robot configuration and environment to navigate. These local planners are eband_local_planner, dwa_local_planner, and teb_local_ that adhere to nav core:: Base Local Planner. For the mobile robot's local planning, we used dwa_local_planner.

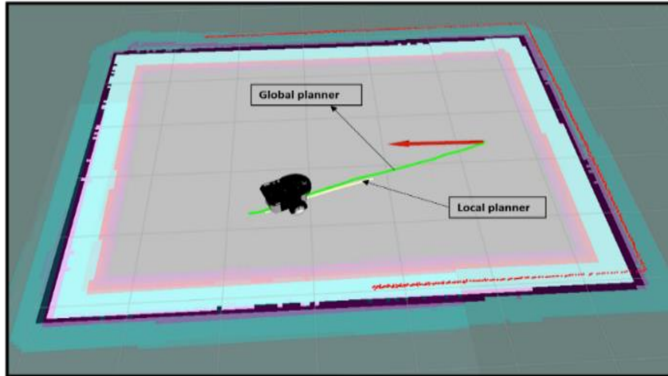


Figure 4-23 Example of a local planner

The configuration of the local planner parameters is more critical, complicated, and delicate than that of the global planner. Table 4.9 from the ROS Wiki [39] lists some of the dwa_local_planner parameters and their applications in planning.

Table 4-9 DWA planner parameters

Dwa_Local_Planner Robot Configuration Parameters	
acc_lim_x (double, default: 2.5)	• The x acceleration limit of the robot in $meter/sec^2$
acc_lim_y (double, default: 2.5)	The y acceleration limit of the robot in
acc_lim_th (double, default: 3.2)	• The rotational acceleration limit of the robot in
max_vel_trans (double, default: 0.55)	• The absolute value of the maximum translational velocity for the robot in m/s
min_vel_trans (double, default: 0.1)	• The absolute value of the minimum translational velocity for the robot in m/s
max_vel_x (double, default: 0.55)	• The maximum x velocity for the robot in m/s
min_vel_x (double, default: 0.0)	• The minimum x velocity for the robot in m/s, negative for backwards motion.
max_vel_y (double, default: 0.1)	• The maximum y velocity for the robot in m/s
min_vel_y (double, default: -0.1)	• The minimum y velocity for the robot in m/s
max_rot_vel (double, default: 1.0)	• The absolute value of the maximum rotational velocity for the robot in rad/s
min_rot_vel (double, default: 0.4)	• The absolute value of the minimum rotational velocity for the robot in rad/s

조선대학교
CHOSUN UNIVERSITY

Dwa_Local_Planner Goal Tolerance Parameters	
yaw_goal_tolerance (double, default: 0.05)	• The tolerance in radians for the controller in yaw/rotation when achieving its goal
xy_goal_tolerance (double, default: 0.10)	• The tolerance in meters for the controller in the x & y distance when achieving a goal
latch_xy_goal_tolerance (bool, default: false)	• If goal tolerance is latched, if the robot ever reaches the goal xy location it will simply rotate in place, even if it ends up outside the goal tolerance while it is doing so.

Dwa_Local_Planner Forward Simulation Parameters	
sim_time (double, default: 1.7)	• The amount of time to forward-simulate trajectories in seconds
sim_granularity (double, default: 0.025)	• The step size, in meters, to take between points on a given trajectory
vx_samples (integer, default: 3)	• The number of samples to use when exploring the x velocity space
vy_samples (integer, default: 10)	• The number of samples to use when exploring the y velocity space
vth_samples (integer, default: 20)	• The number of samples to use when exploring the theta velocity space
controller_frequency (double, default: 20.0)	• The frequency at which this controller will be called Hz.

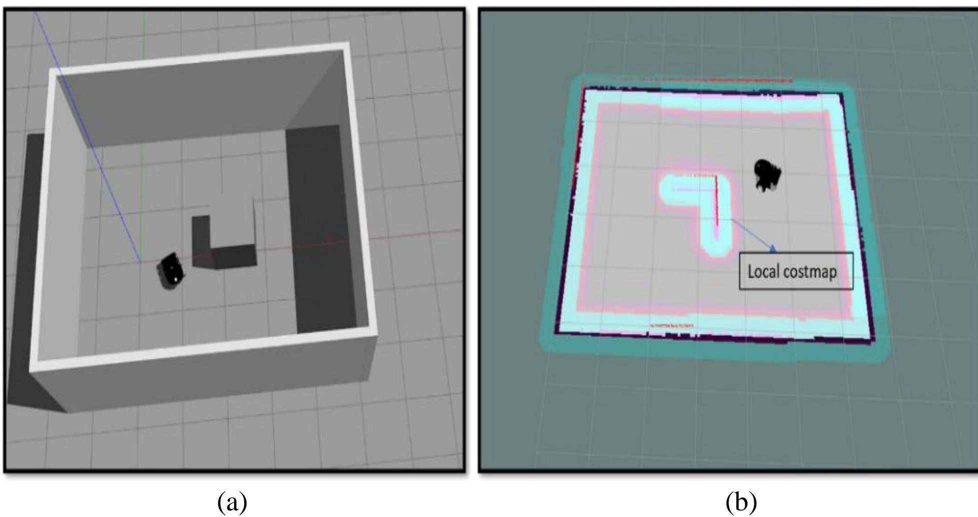
Dwa_Local_Planner Trajectory Scoring Parameters	
path_distance_bias (double, default: 32.0)	• The weighting for how much the controller should stay close to the path it was given.
goal_distance_bias (double, default: 24.0)	• The weighting for how much the controller should attempt to reach its local goal, also controls speed.
occdist_scale (double, default: 0.01)	• The weighting for how much the controller should attempt to avoid obstacles.
max_scaling_factor (double, default: 0.2)	• The absolute value of the velocity at which to start scaling the robot's footprint in m/s

조선대학교
CHOSUN UNIVERSITY

The impact of the aforementioned parameters on local planning will also be discussed in the results section later. The DWA planner relies on the local costmap that provides information about obstacles. Due to this, for the optimal behavior of the DWA local planner, fine-tuning the parameters for the local costmap is critical.

Local Costmap

A local costmap is created by inflating obstacles sensed in real-time by the robot's sensors. Given a width and a height for the costmap defined by the user that keeps the robot in the center of the costmap while it is moving throughout the environment, it drops information from the map about obstacles as the robot moves. We have created a small environment on Gazebo with an obstacle to illustrate the local costmap (shown in Figure 4-24) around the obstacle. This environment also is used in this chapter.



(a) (b)

Figure 4-24 Example of local costmap.

Because the global and local cost maps do not have the same property, they have different parameters. Table 4-10 shows some of the local costmap parameters and common costmap parameters for both global and local cost maps.

조선대학교
CHOSUN UNIVERSITY

Table 4-10 Local costmap and common costmap parameters

Local Costmap Parameters	
global_frame (string, default: "/odom")	• The global frame for the costmap to operate in the local costmap
robot_base_frame (string, default: "base_link")	• The name of the frame used for the base link of the robot.
rolling_window (bool, default: true)	• Whether or not to use a rolling window version of the costmap. If the static_map parameter is set to true, this parameter set to false.
Update_frequency (double, default: 5.0)	• The frequency in Hz for the map to be updated.
width (int, default: 10)	• The width of the costmap
height (int, default: 10)	• The height of the costmap
plugins	• It is the sequence of plugin specifications, one per layer. Each specification is a dictionary with a name and type fields. The name is used to define the parameter namespace for the plugin.

Common Costmap parameters	
footprint (list, default: [])	• The footprint is used to calculate the radius of the inscribed and circumscribed circles, which are then used to inflate obstacles to fit the robot.
Robot_radius (double, default: 0.46)	• The robot's radius in meters. This parameter is only set for circular robots; for all other types of robots should use the "footprint" parameters.
Layer parameters (Each layer has its own parameters)	• Static layer: responsible for providing the static map to the costmap that requires it (usually global costmap) • Obstacle layer: it is used for marking and clearing operations. • Inflation layer: responsible for performing inflation in each cell with an obstacle.

The concepts and implementation of ROS algorithms discussed with the robots in this section will be implemented in the simulation and experiment in the following chapter.

조선대학교
CHOSUN UNIVERSITY

5. RESULTS AND DISCUSSIONS

In this chapter, autonomous navigation to achieve the goal pose (position and orientation) by avoiding obstacles with the pre-constructed map, and the development of a Graphical User Interface (GUI) for waypoint navigation of mobile robots will be discussed. To test the abovementioned algorithms, we divided the result into two parts: simulation results and experimental results. We will also investigate the parameters used and their effects on mapping, localization, and path planning quality in the experimental result section.

5.1. Simulation Results

We use Gazebo platform to create a simulation environment for the experiment depicted in Figure 5-1. The virtual environment has real physical properties, and the simulation results are a close match to the real world.

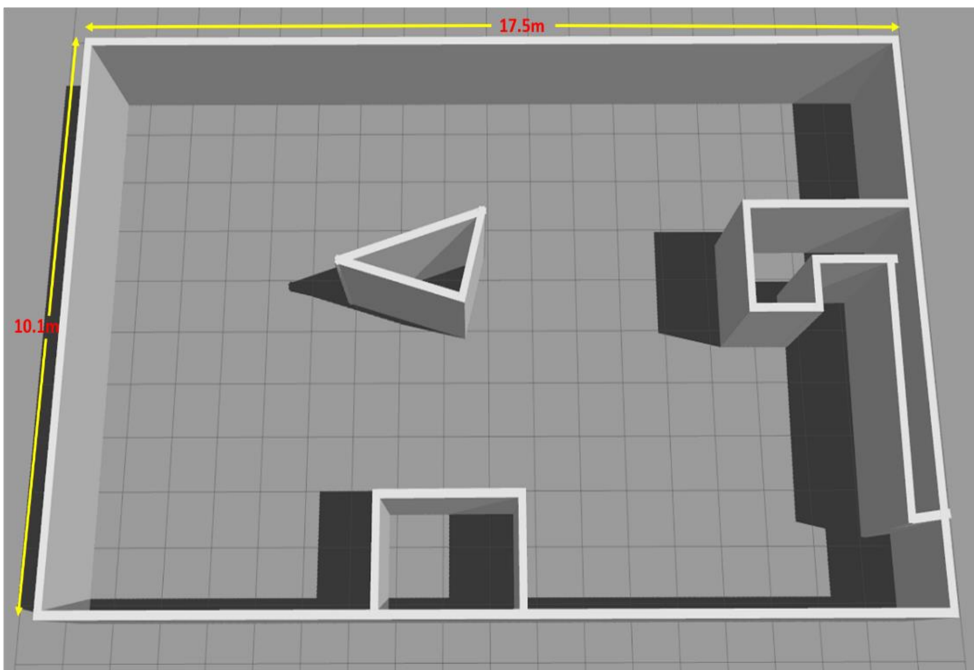


Figure 5-1 Simulation environment constructed on Gazebo.

조선대학교
CHOSUN UNIVERSITY

(a) Mapping of the Virtual Environment

The simulation experiment below was carried out in accordance with the virtual environment. The first phase entails integrating the MRP NRLAB02 mobile robot platform with the hokuyo 2d lidar scanner (hokuyo.dae mesh was used to render it) as illustrated in Figure 4.8 under the ROS Navigation Structure section. We tested ROS gmapping to map the unknown environment after we interfaced with the robot's URDF and lidar sensor by taking lidar and wheel encoder information and using the teleoperation node that receives keyboard inputs to send velocity commands to the robot's */nrlab/cmd_vel* topic. The mapping algorithms are tested using the ROS robot system's simulation platform, and the simulation results are shown in Figure 5-2.

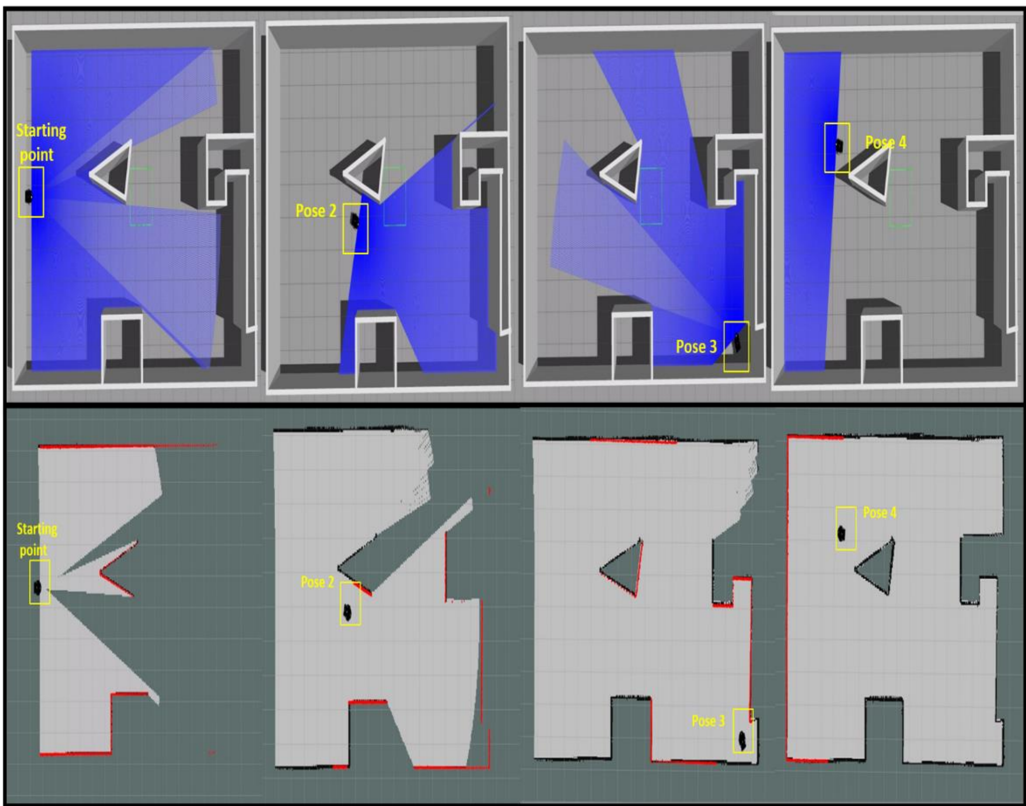


Figure 5-2 Occupancy Grid Map of the virtual environment on Gazebo and Rviz

Figure 5-1 depicts the robot's behavior during the simulation process while using SLAM gmapping. The robot is positioned in the center of the simulation environment's left wall, and the

조선대학교
CHOSUN UNIVERSITY

lidar data is highlighted in red, with the blue ray on Gazebo representing the lidar's ray by activating the lidar visualization plugin on the URDF of the hokuyo sensor. The simulated hokuyo lidar has 180^0 horizontal fields of view and a maximum detection range of 15m, which we cropped to 10m for mapping the virtual environment. The area, when scanned by lidar, changes from light gray to white and black when the robot is moving through the environment. When the robot moves through the environment, the scanned area changes from light gray to white and black until the entire map is completed. The light gray shade represents the absence of information about the area, free space is represented by white shade, and the detection of obstacles in the environment is represented the black lines by the black lines. Table 5-1 shows the coordinate values for the four poses in Figure 5-2, with a position offset of $(-0.089m, -0.1m, 0.0012rad)$ from the origin of the robot to the coordinate system of the map at $(0,0,0)$. Figure 5.3 depicts the communication between nodes and topics, while Figure 5-4 depicts the transformation tree of robot links, lidar sensor, and parent frame id map.

Table 5-1 Some positions and orientations of the mobile robot during the mapping operation.

	X-Position (m)	**Y-Position (m)**	**Yaw-Orientation (rad)**
Starting Point	-0.089	-0.1	0.0012
Pose 2	4.325	-0.93	-0.46
Pose 3	14.87	-5.62	2.27
Pose 4	3.17	0.77	3.12

조선대학교
CHOSUN UNIVERSITY

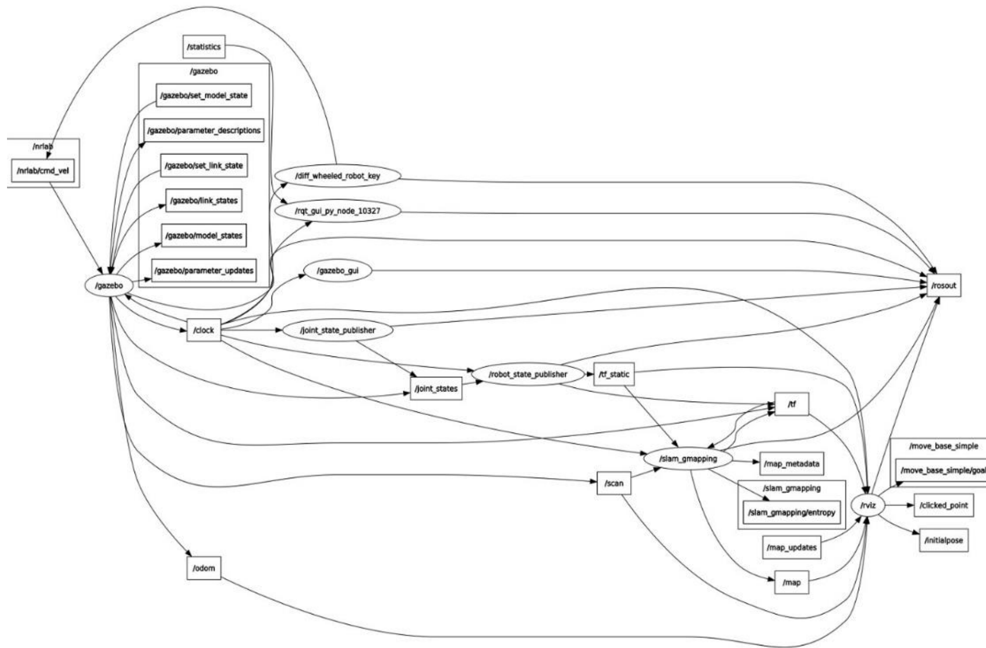


Figure 5-3 Communication between nodes and topics using rqt_graph

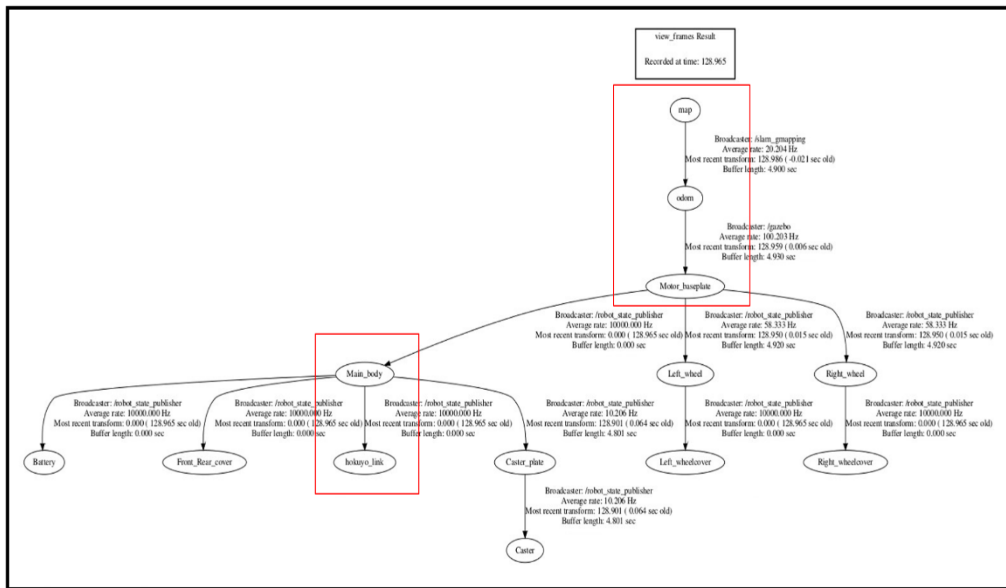


Figure 5-4 Transformation tree after launching the mapping algorithm.

According to the simulation results, a compatible map of the environment with the same dimensions and features as the virtually constructed environment was created. The optimization

조선대학교
CHOSUN UNIVERSITY

of several *slam_gmapping* node parameters has been done. In the simulation, the built-up map of the environment will be used for the mobile robot's autonomous navigation. The effect of mapping parameters will be discussed in the experimental section.

(b) Navigation of Mobile Robot in the Constructed Map

After creating the static map, the navigation task can be carried out. Before the robot can move from one point to the next, it must first locate itself on the map. This is accomplished by the AMCL node, which, as explained in section 4.3.2, estimates the position of the robot using particles.

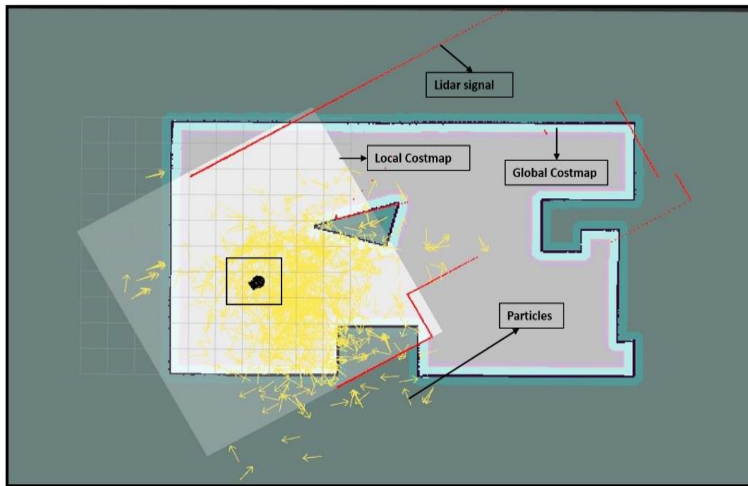
In the simulation section, we tested the navigation of the MRP_NRLAB02 mobile robot from one point to the target goal in two cases: navigation by avoiding static and dynamic obstacles inside the grid map. In the practical experimental section, the parameters and their effect on navigation, as well as the operation of the algorithms used for each component of the navigation system, will be explained in detail.

i. Navigation of the simulated robot inside the map with static obstacles

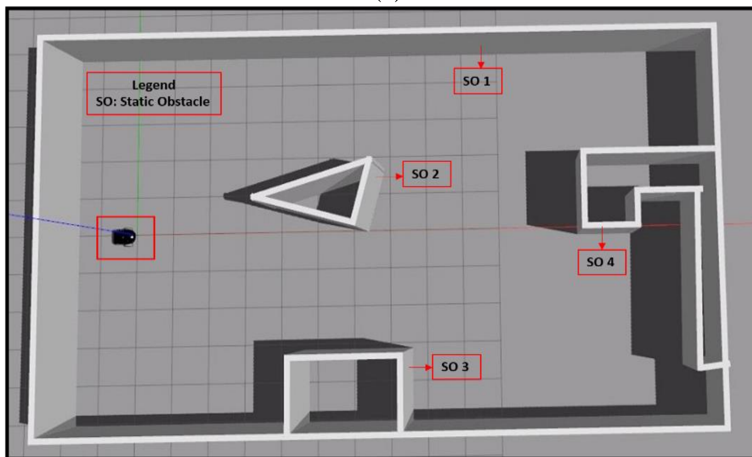
The map which is created from SLAM gmapping has been loaded via the map server package from the path where the map is saved to use to autonomously navigate the robot in the environment.

Figure 5-5 (a) on Rviz shows the laser signal, the loaded map, the global costmap, the local costmap, particles to estimate the robot's pose, and static obstacles in the static map before navigation begins. Figure 5-5 (b) depicts the environment and the robot's true pose for comparison with the robot's internal computation. Figure 5-5 (a) demonstrates that the particles distributed around the map show the robot uncertainty about its true position and orientation when compared to the one shown in the Gazebo environment (b).

조선대학교
CHOSUN UNIVERSITY



(a)



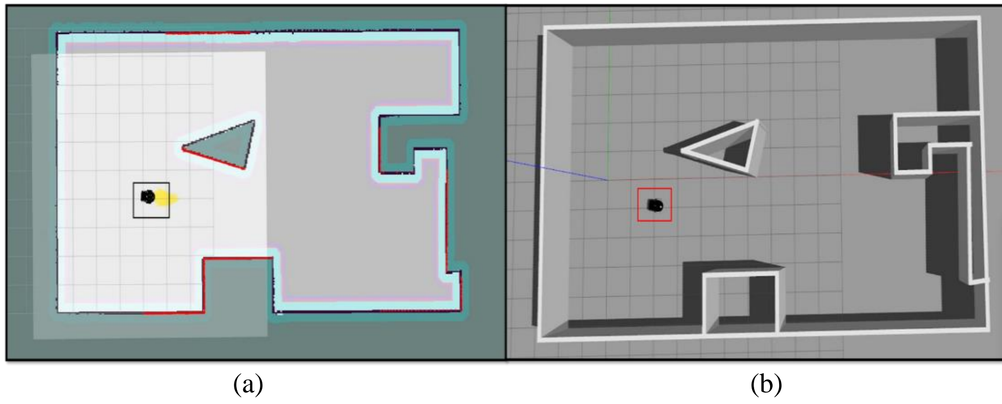
(b)

Figure 5-5 Preparing the simulated mobile robot for navigation on Rviz (a) and Gazebo (b) on the static obstacles in the environment.

The laser signal does not match the environment, as shown in Figure 5-5 (a), because the internal pose state of the robot and the real robot (Figure 5-5 (b)) have some position and orientation offsets, which also dispersed the particle's pose to guess the robot's location. This can be corrected by teleoperating the robot around the environment to collect more information from the laser and the map depicted in Figure 5-6, or we can use Rviz's *2D pose estimator* to estimate the exact pose of the robot in the environment; in our case, both work perfectly fine. The

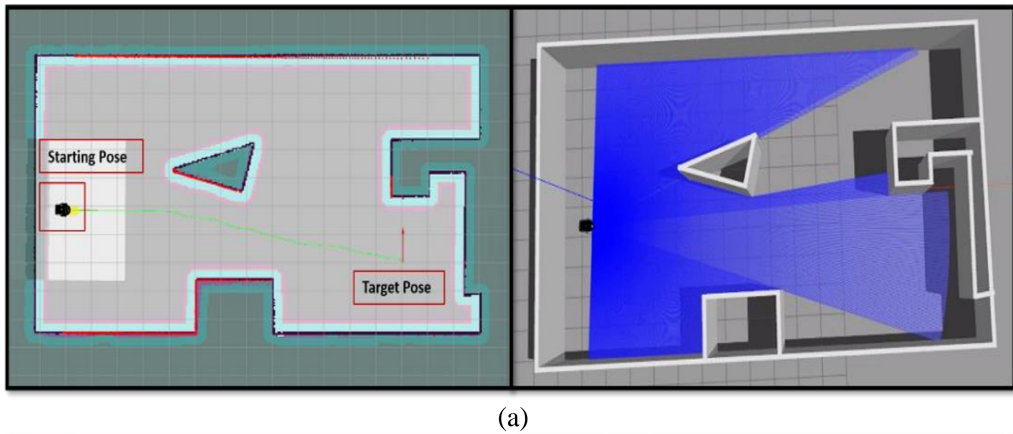
조선대학교
CHOSUN UNIVERSITY

constructed simulated environment on Gazebo (Figure 5-5 (b)) depicts a collection of static obstacles in the environment ready for autonomous navigation

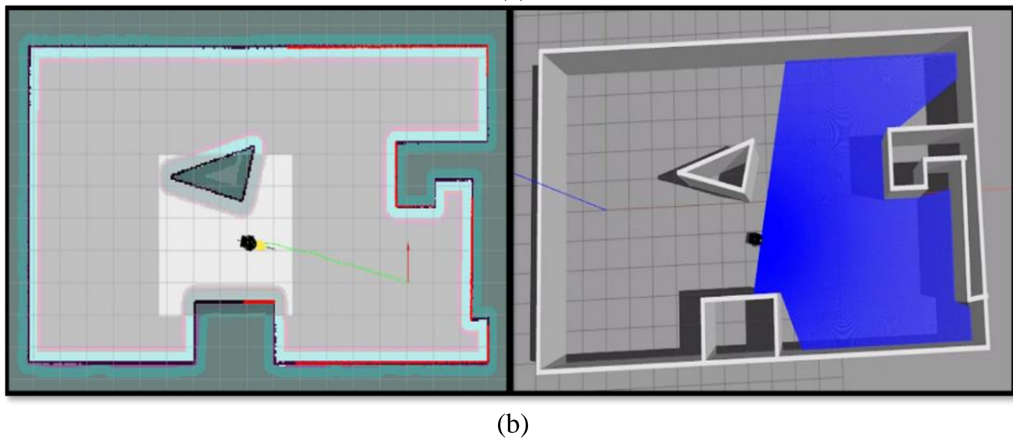


(a) (b)

Figure 5-6 Correction of the robot's pose for navigation on Rviz (a) and Gazebo (b) following a small robot motion

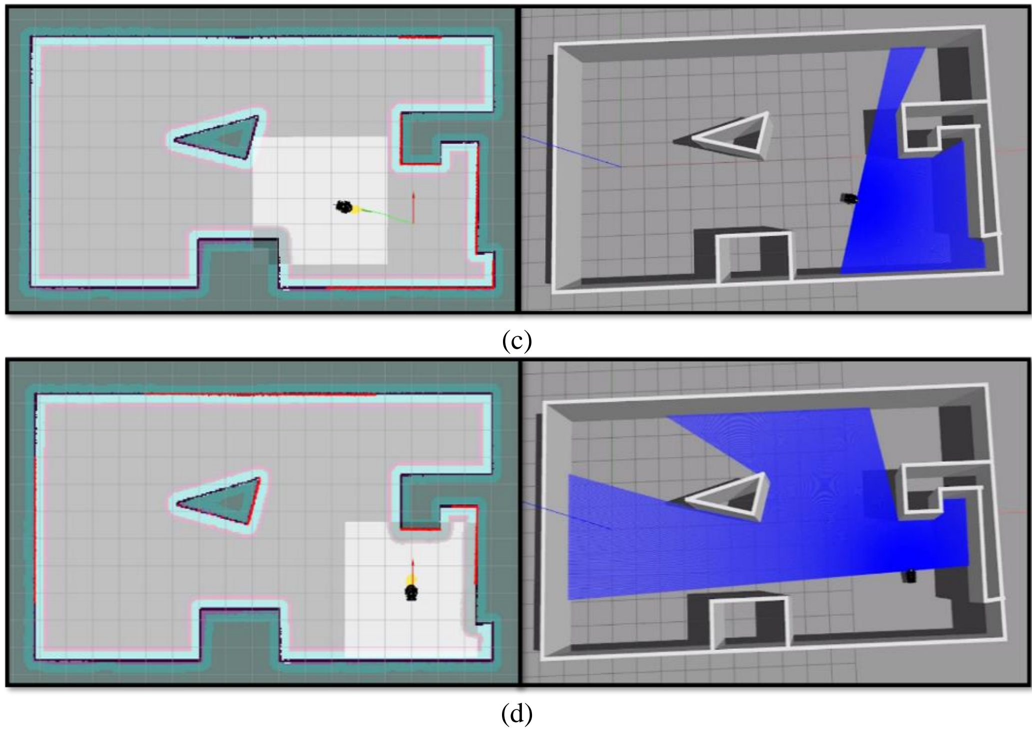


(a)



(b)

조선대학교
CHOSUN UNIVERSITY



(c)

(d)

Figure 5-7 Autonomous navigation of mobile robot in a simulated environment with static obstacles of the environment.

Table 5-2 Comparison of target goal pose and robot estimated pose with their respective errors of static obstacles in the environment ready for autonomous navigation.

Coordinates	**Robot's Starting Pose (meters)**	**Target Goal Pose (meters)**	**Robot's estimated pose after navigation (meters)**
X	-0.154	12.522	12.433
Y	-0.634	-2.438	-2.4297
Yaw (rad)	-0.0447	1.5708	1.565
Yaw (degree)	-2.56	90	89.67
Absolut X-Distance Error (meters)	0.089		
Absolute Y-Distance Error (meters)	0.0083		
Absolute Orientation Error (rad)	0.0058		

조선대학교
CHOSUN UNIVERSITY

Figure 5-7 depicts the outcome of the path planning for both global and local paths through the static obstacles inside the static map. The red arrow in the image (a) indicates the navigation target position and orientation, and its coordinate values are listed in Table 5-2. Pictures (b) and (c) depict path planning and navigation execution while the robot is moving to the target, and picture (d) depicts when the robot achieves the target point, which is also indicated by the smaller yellow patch denoting the convergence of particles to the final pose. The communication of the nodes and topics after launching the amcl and move_base node is shown in Figure 5-8.

Figure 5-8 Communication of node and topic of the navigation system.

조선대학교
CHOSUN UNIVERSITY

ii. Navigation of the simulated robot inside the map with dynamic obstacles

To test the navigation system in the simulated environment with dynamic obstacles, we added dynamic obstacles such as a standing person, a walking person, and a box with a $1m\ X\ 1m$ size as shown in Figure 5-9 on Gazebo to resemble the real-world scenario.

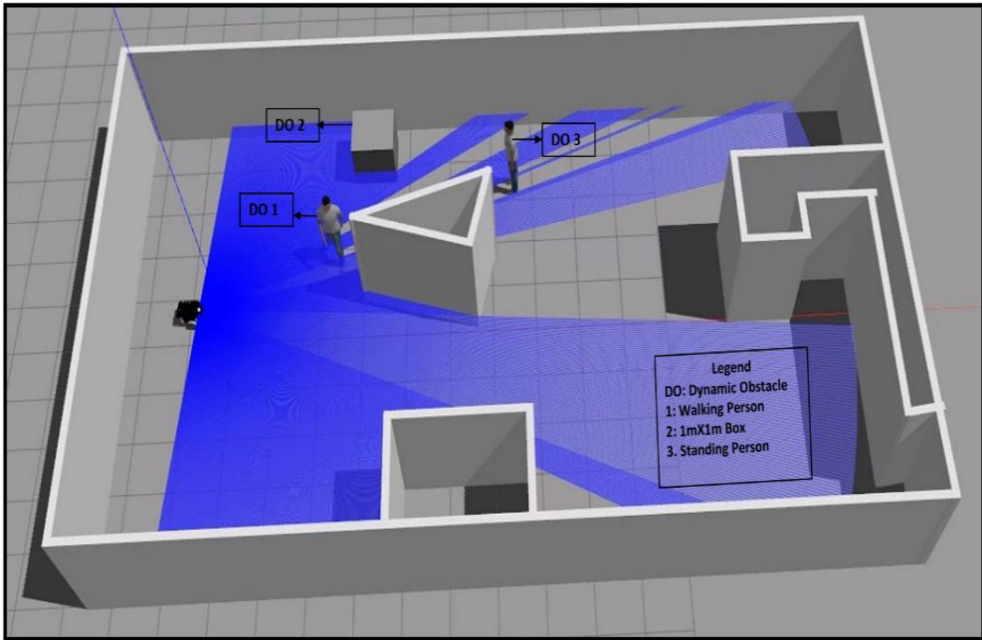


Figure 5-9 Dynamic obstacles in the simulated environment.

The dynamic obstacles are scattered throughout the environment. When the robot plans its trajectory to reach the required goal point, the global costmap is created by inflating the obstacles on the navigation stack's map, as shown in Figure 5-7; however, if there are dynamic obstacles, as shown in Figure 5-9, that are not known by the previously stored map data, the local costmap inflates obstacles detected by the robot's lidar sensor (Figure 5-10) in real-time, hence the local planner can determine the trajectory.

Figure 5-10 shows how the three new obstacles in the environment affect the global trajectory of the global planner generated by the Dijkstra algorithm because the new obstacle is detected by the laser so that the path sent to the local planner will execute each segment of the

조선대학교
CHOSUN UNIVERSITY

global planner and once the new obstacles are inflated by the local costmap, the inflated obstacles are avoided and a new path is generated to reach the required goal.

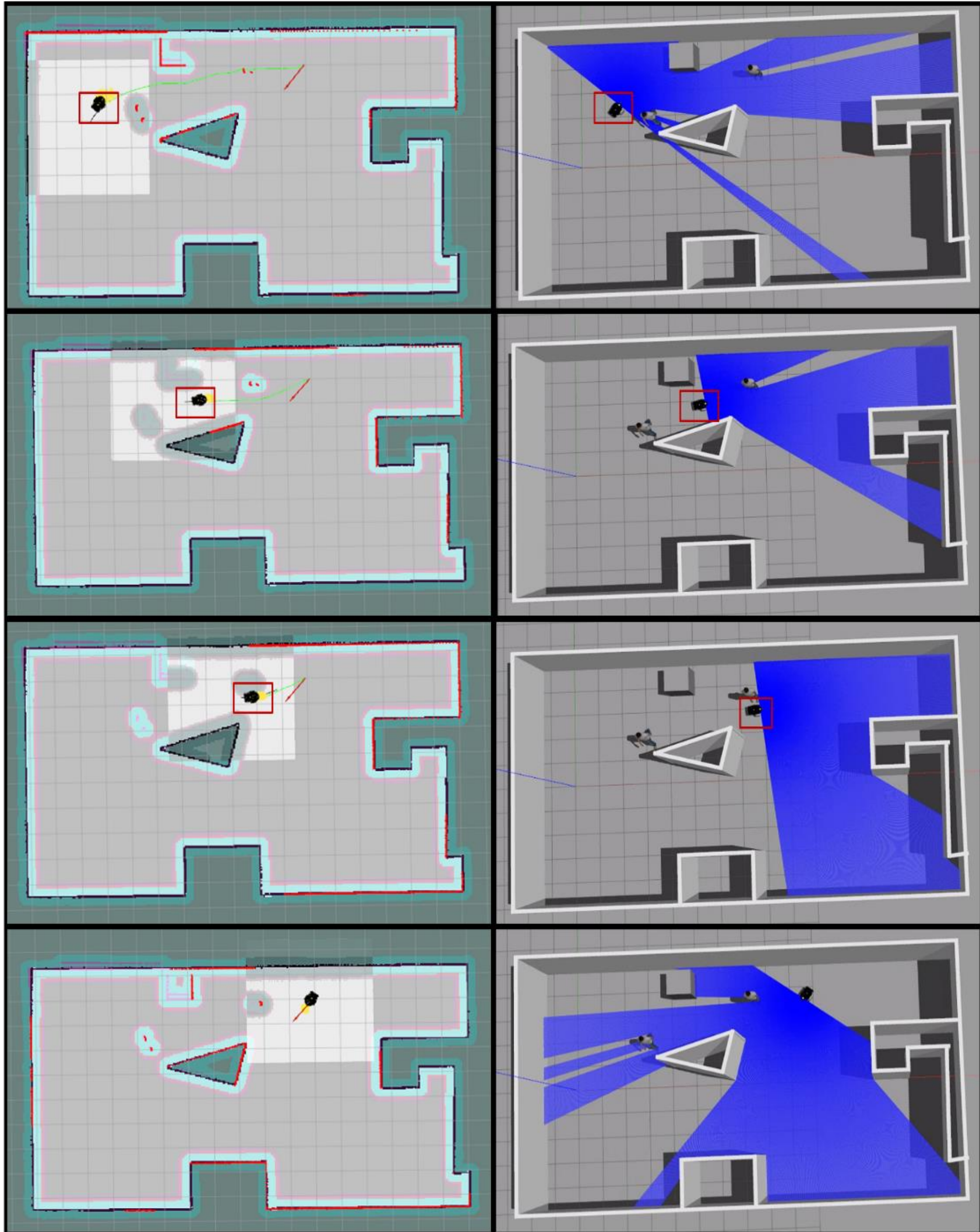


Figure 5-10 Autonomous navigation of a mobile robot in a simulated environment with dynamic obstacles of the environment.

조선대학교
CHOSUN UNIVERSITY

The green line in the above figure represents the path that the robot takes to reach the goal pose given by Rviz's 2D Nav Goal tool, and the red arrow represents the target goal pose that the robot must address, and the target pose values and the robot estimated pose in the environment is shown in Figure 5-11.

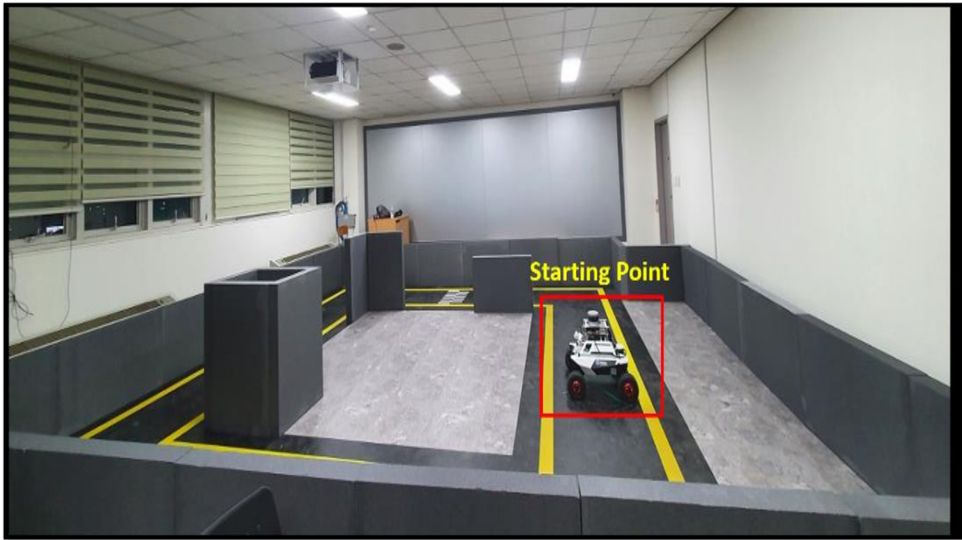
(a) (b)

Figure 5-11 The target position and orientation (a) and the estimated pose (b) of the robot in Figure 5.10

5.2. Experimental Results

The final set of results is generated and observed using the actual environment setup shown in Figure 5-12. We carried out the experiments in two real-world environments that we divided into two scenarios (Scenario 1 and Scenario 2), and we used the Rviz platform to analyze the internal state of the robot and compare it to the actual robot status.

조선대학교
CHOSUN UNIVERSITY



(a)

(b)

Figure 5-12 Experimental indoor environments for navigation of a mobile robot

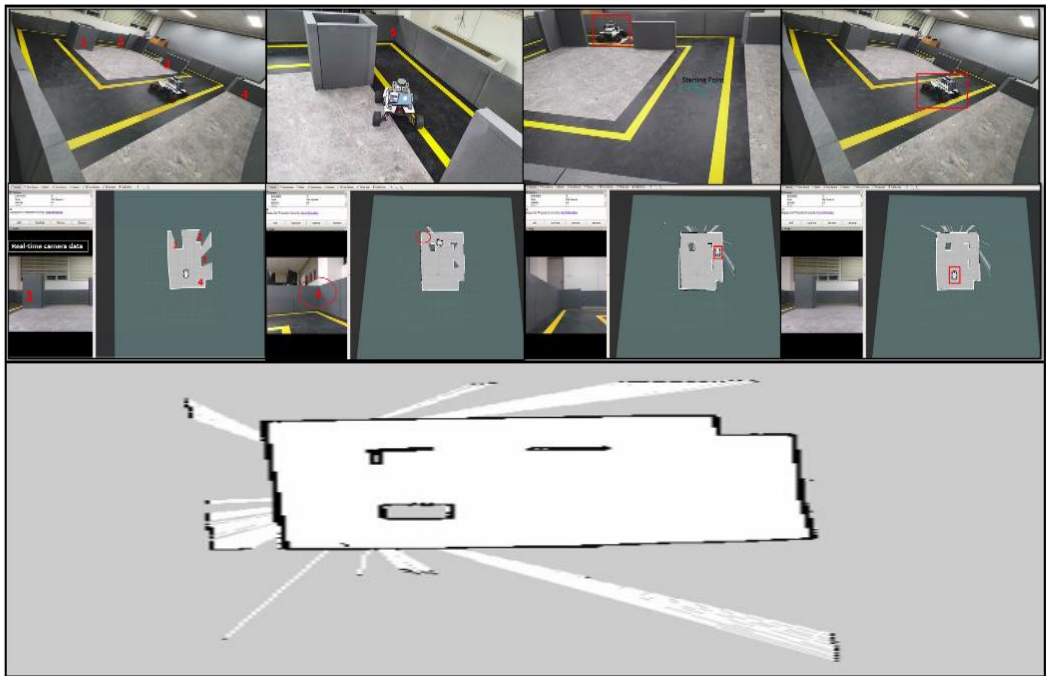
(a) The first indoor environment used in ***Scenario 1*** is our lab experiment room with an area of approximately $34m^2$.

(b) The second indoor environment used in ***Scenario 2*** is the 6th floor of the IT-Convergence building at Chosun University, South Korea with an area of $114.25m^2$, the collection of images is taken from different side views.

조선대학교
CHOSUN UNIVERSITY

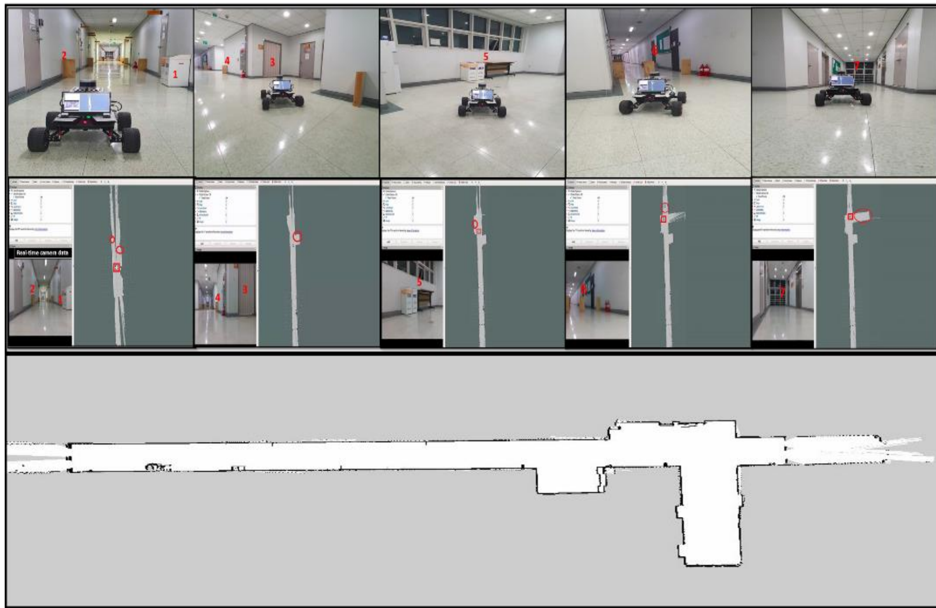
(a) Mapping of the Experimental Environments

The gmapping algorithm's experimental performance was evaluated in the two scenarios depicted in Figure 5-12. Figure 5-13 depicts the mapping result obtained from the experiments while the Scout Mini mobile robot is teleoperated around the two environments and the final mapping outputs, as well as the real-time data visualization from the realsense camera, which is used to compare the obstacles detected by the grid map with the robot's actual position and attitude. The experimental results for the two scenarios showed the creation of a compatible map of the actual environment with the same dimensions and features. To demonstrate this, the numbers 1 through 6 are marked on the two environments in Figure 5-13 with their respective maps.



(a)

조선대학교
CHOSUN UNIVERSITY



(b)

Figure 5-13 Mapping of the actual environments for Scenarios 1 (a) and 2 (b), with comparisons of environmental features and the constructed map on Rviz.

During the experiment, the gmapping algorithm uses a static transformation between the laser's base link frame (with link name “velodyne”) and the robot base link frame (with link name of base_link) to generate three-dimensional measurement data with a topic of ***/velodyne_points***, which is changed to 2D laser scan data using the ROS package called ***pointcloud_to_laserscan*** to publish a ***/scan*** topic with a message of ***sensor_msgs/msg/LaserScan*** from the point cloud. The data from the wheel encoder is used to provide odometry data to the *slam_gmapping* node with the topic ***/odom***, which is then interfaced with the lidar topic to publish the ***/map*** topic to generate the map of the environment, as shown in Figure 5-14. Additionally, several parameters of the *slam_gmapping* node has been optimized, and some of the major parameters and their values for our experimental case are shown in Table 5-3. The created map is used by localization and path planning algorithms to complete real-time robot navigation.

조선대학교
CHOSUN UNIVERSITY

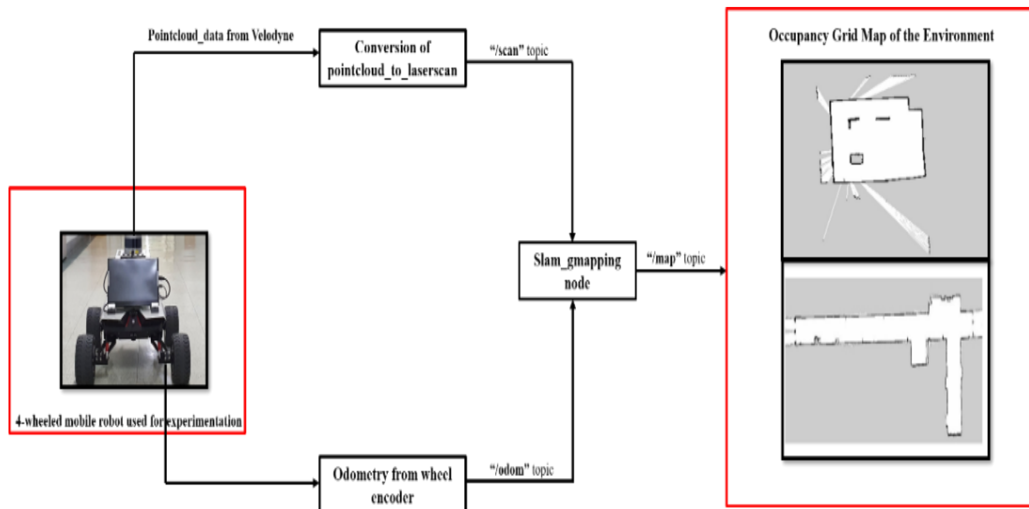


Figure 5-14 Block diagram representation of the mapping algorithm of the experiment

Table 5-3 Some gmapping parameter values used for the experiment

	Values
map_update_interval	0.5
maxUrange	20.0
maxRange	100.0
particles	100
delta	0.05
linearUpdate	0.3
angularUpdate	0.5

(b) Localization of a Mobile Robot Inside the Environment

The map generated in the previous section from actual environments is used to localize the robot using the AMCL method, just like the simulation results. In the experimental analysis of the localization, we have divided it into two cases for the two scenarios. Table 5-4 shows some of the major parameters used in the AMCL ROS package for the two cases.

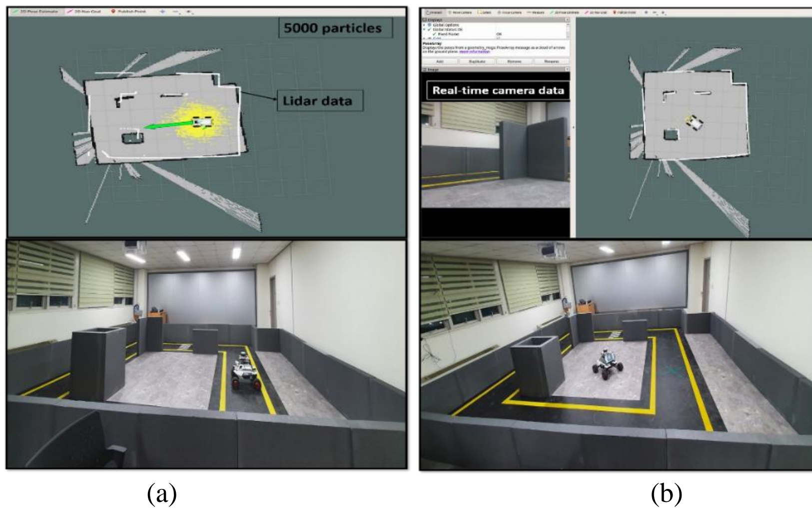
조선대학교
CHOSUN UNIVERSITY

Table 5-4 Some of the AMCL parameters for the two cases

	Case-1 parameters	Case-2 parameters
min_particles	500	5
max_particles	5000	10
odom_alpha1	0.15	0.15
odom_alpha2	0.15	0.15
odom_alpha3	0.3	0.3
odom_alpha4	0.3	0.3
update_min_d	0.25	0.25
update_min_a	0.2	0.2

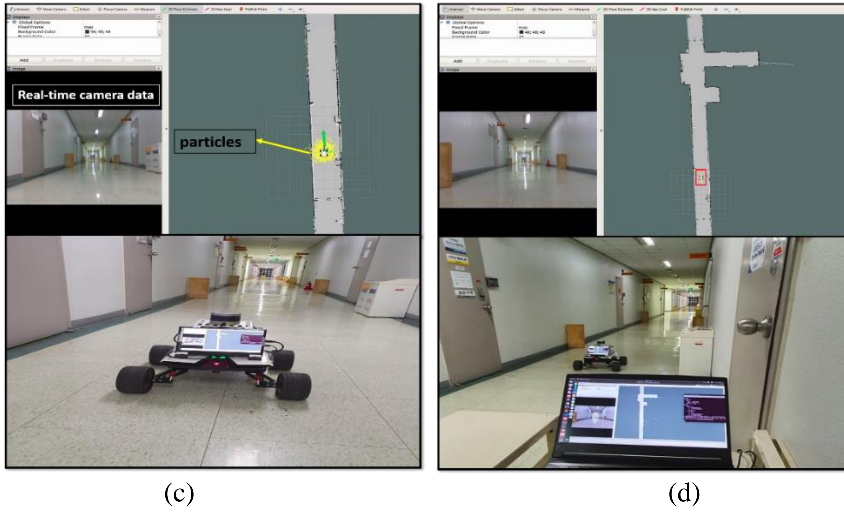
i. Localization of the mobile robot with Case-1 parameters

For the case-1 experiment, we used 500 *min_particles* and 5000 *max_particles* for the estimation of the robot's pose in the environment. To localize the robot, we used two methods: manual localization and global localization. Figure 5-15 depicts the explanation for Scenarios 1 and 2 for manual localization, while Figure 5-16 represents global localization for both scenarios.



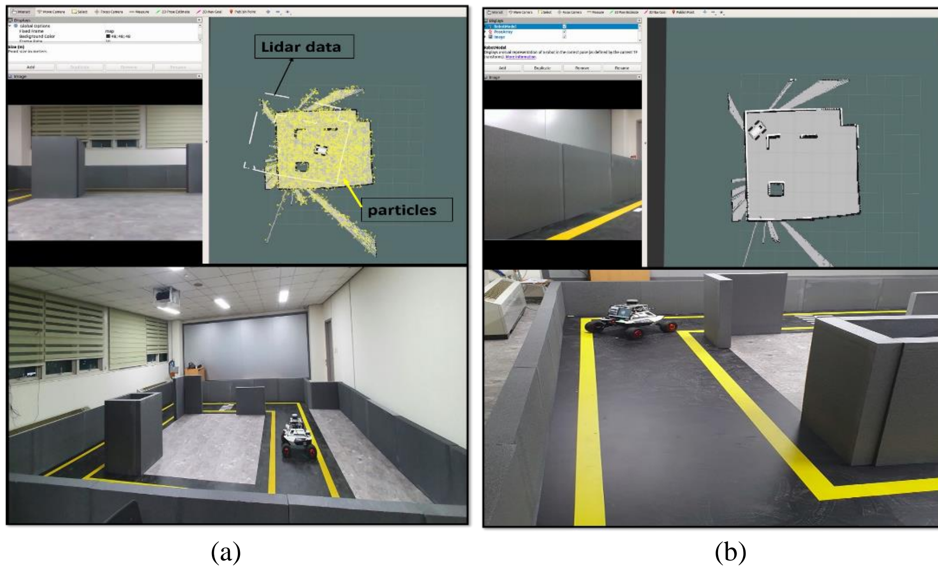
(a) (b)

조선대학교 CHOSUN UNIVERSITY



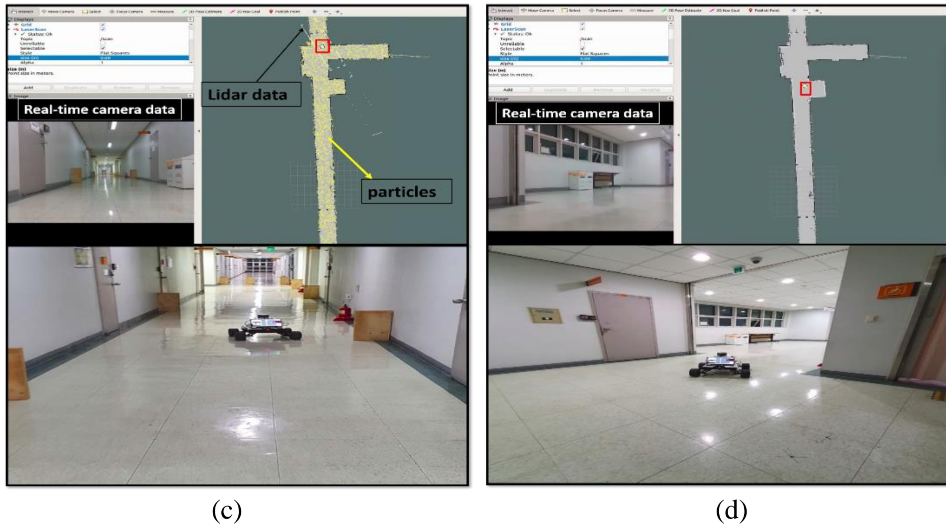
(c) (d)

Figure 5-15 Manual localization of mobile robot for Scenario 1 and Scenario 2 using 2D pose estimator tool (a), (c) The yellow color arrow of the particles used to estimate the robot pose in Scenarios 1 and 2 depicts the uncertainty in robot position at the beginning of AMCL. The green arrow represents the robot's manual pose estimation in the environment using the *2D pose estimator* tool of Rviz. (b), (d) Represents the condensation of particles after the robot translates 1.6 meters in the positive x-direction for Scenario 1 and 3.29 meters forward in Scenario 2.



(a) (b)

조선대학교
CHOSUN UNIVERSITY



(c) (d)

Figure 5-16 Global localization of a mobile robot for Scenario 1 and Scenario 2. (a), (c) After calling AMCL's *global localization* service, the particles are distributed uniformly (yellow arrows) across the entire map, with equal guesses about the robot's pose. Because the robot pose estimation and the actual robot poses differ, the lidar measurement (white line) does not match the map. (b), (d) The convergence of particles after the robot moves and when the filter updates its belief due to the motion, the measurements were projected from the robot's pose point of view. For Scenario 1 the distance the filter took to converge is 4.15 meters and 21.38 meters for Scenario 2.

ii. Localization of the mobile robot with Case-2 parameters

In the case-2 experiment, we estimated the robot pose in the environment using 5 min particles and 10 max particles, as shown in Figure 5-17 which depicts the filter's pose estimation after moving the robot around the environment in Scenarios 1 and 2. The results show that even if the robot is teleoperated around the environment for a long time, it may be unable to localize itself. This is because there are no particles close to the robot's true position during the particle filter's correction and resampling step. Figure 5-19 depicts the operation of the particle filter, which demonstrates the internal analysis of the algorithm used to localize the robot.

조선대학교
CHOSUN UNIVERSITY

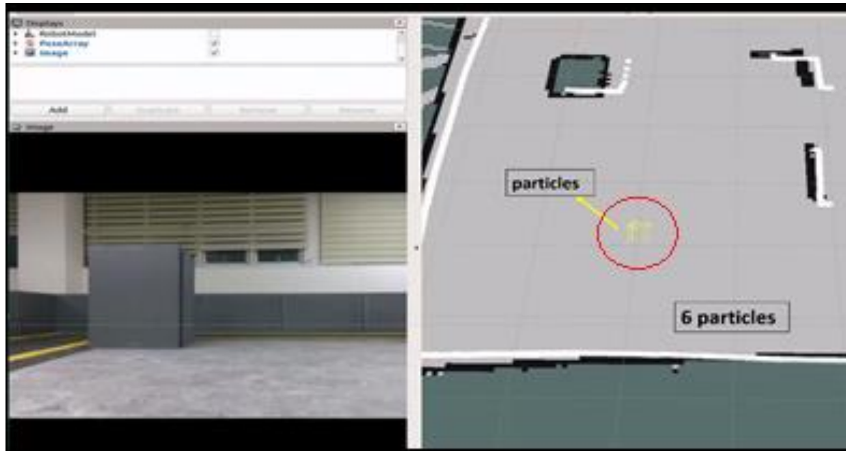


Figure 5-17 Particles used for Case-2 parameters.

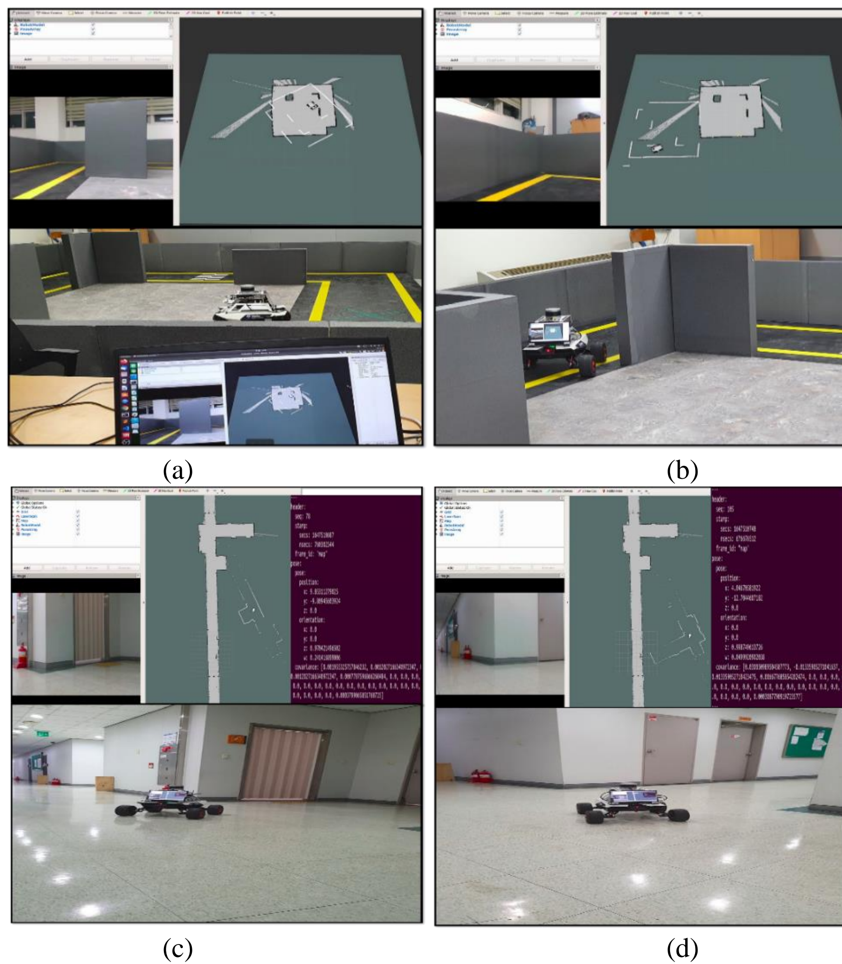


Figure 5-18 The effect of small particles on the localization of the robot.

조선대학교
CHOSUN UNIVERSITY

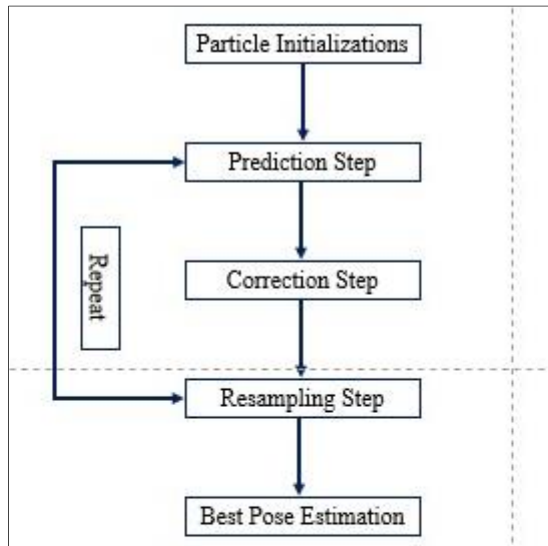


Figure 5-19 Structure of particle filter.

- **Particle Initializations:** each particle is a belief of where the robot can be located, and each particle has a position (x,y) and orientation to keep the detail of the particle's position guess. Each particle also has an associated weight. This is shown in Figure 5-15. From the computational perspective, adding more particles can increase the filter's precision, but the number of computations required in the localization process is also much higher.
- **Prediction step:** when the robot moves, all the particles move with the same motion. To account for process noise, the prediction step adds multivariate Gaussian noise to each particle motion, causing particle dispersion across the map. This distribution of particles is essential to ensuring that the robot's true pose is covered by particles, regardless of the robot's noise motion. During the prediction step, particles are propagated forward as determined by the noisy odometry motion model.
- **Correction step:** sensor measurements are processed in this step. It corrects the state for the next filter iteration based on sensor measurements. It processes incoming measurements, compares them to the measurements of the particles on the map, and then prioritizes particles with the lowest error between measurements and the map. The

조선대학교
CHOSUN UNIVERSITY

degree of correspondence between the distances obtained by the real laser scan and the distances calculated by each particle's expected distance measurement determines the likelihood of a particle being near the robot's ground truth pose. During this process, a weight is assigned to each particle. The particle with the highest weight has been used to estimate the robot's position, while particles with low weights are discarded.

- **Resampling step:** after the correct step, the weights of the particles have changed. Resampling is the process of replacing the particles with small weights by others with high probability poses. Without the resampling step, particles would remain spread out over the map, without really making use of the information obtained by the measurements. Resampling helps the particles to condense to the true state of the robot.

During the experiment, the particle filters took roughly 10 to 15 seconds in Scenario 1, and 30 to 45 seconds in Scenario 2. We have observed that increasing the number of particles will improve the amcl's pose estimation performance when combined with a good translation and orientation noise parameter (odom_alphas 1 through 4) to improve the robot's localization performance. These parameters define how much noise is expected from the robot's motions as it navigates inside the map.

(c) Autonomous Navigation of Mobile Robot in the Actual Environments

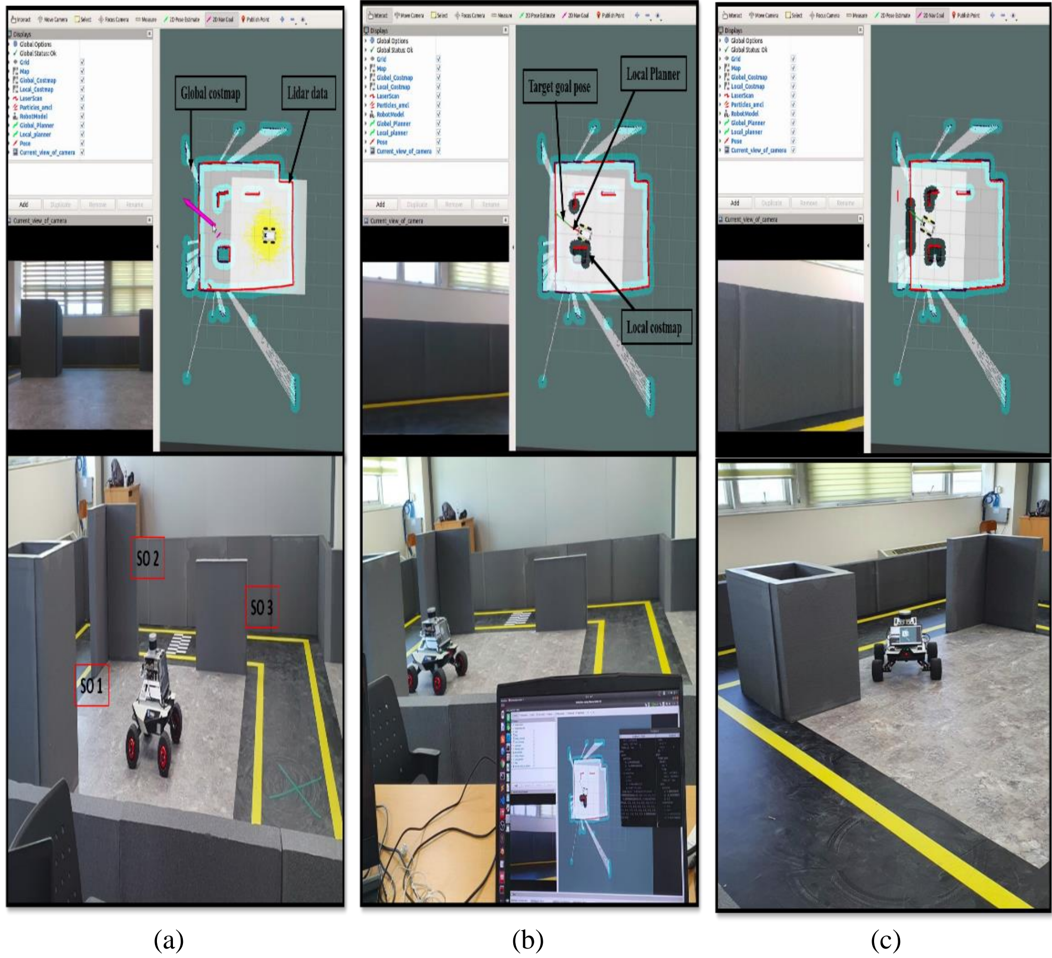
Once the map is generated, the robot can determine its position in the environment in relative to the global map frame and perform path planning and obstacle avoidance. In the experiment section, we have tested the navigation performance for the static and dynamic obstacles in the real-world environment for both Scenarios.

i. Navigation of mobile robot inside the actual environment with static obstacles

The Scout Mini mobile robot's navigation in the static map with static obstacles inside the environment is inflated by the global costmap, as shown in Figure 5-20, which represents places in a grid of cells where the robot is safe to be. When the goal is received by the move_base

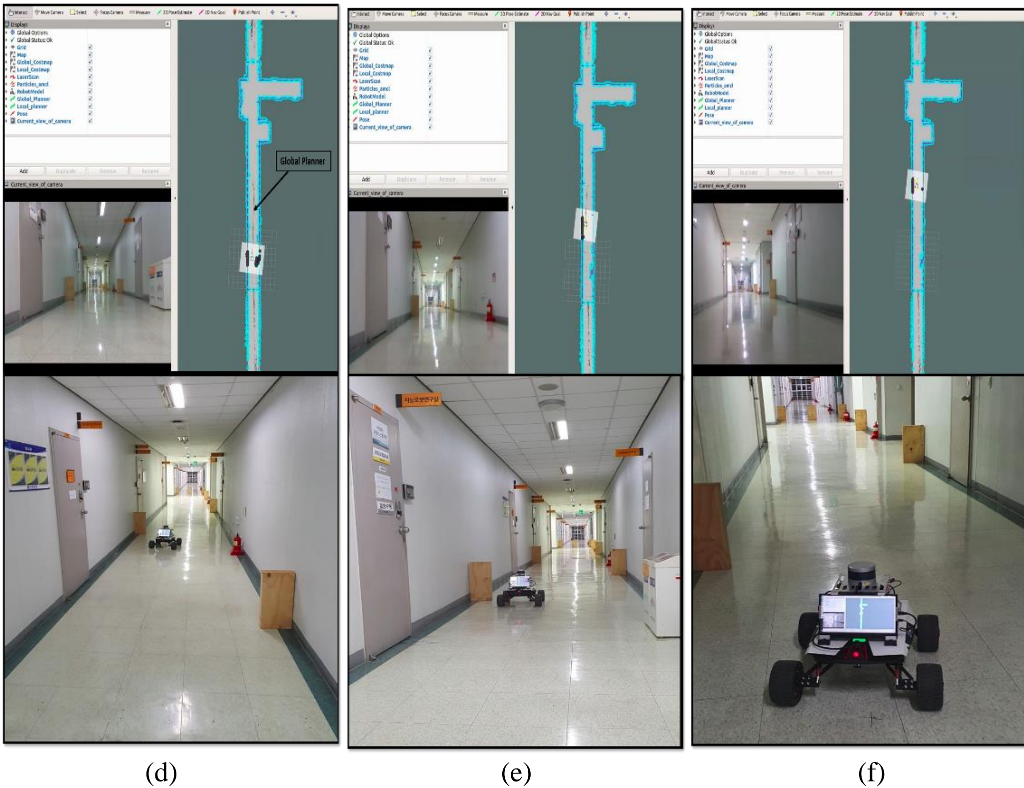
조선대학교 CHOSUN UNIVERSITY

node in the static environment, the global planner shown in Figure 5-21 is in charge of calculating a safe path to the goal pose that does not consider the readings performed by the robot lidar sensor while moving.



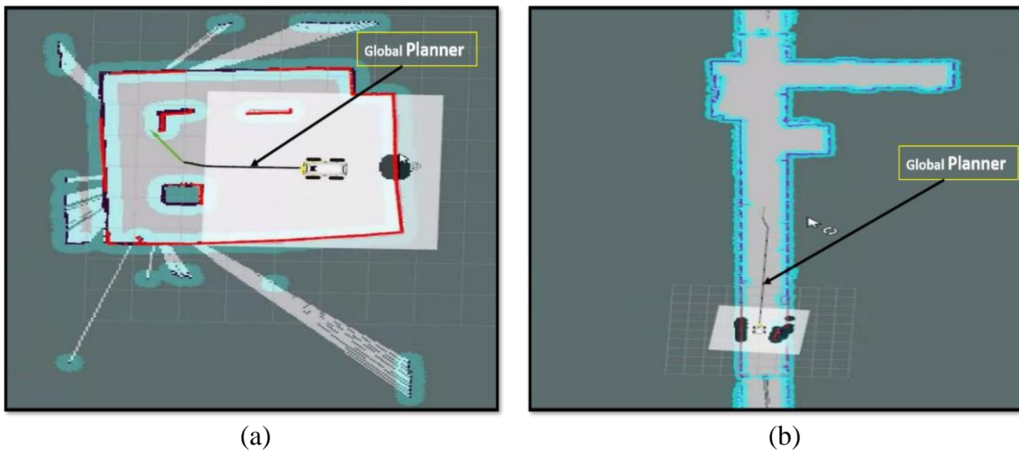
(a) (b) (c)

조선대학교
CHOSUN UNIVERSITY



(d) (e) (f)

Figure 5-20 Autonomous navigation of a mobile robot in the actual indoor environment of the static map without dynamic obstacles for Scenario 1 and Scenario 2.



(a) (b)

Figure 5-21 Global planner of the robot in the static map of the environment.

Figure 5-20 shows the operation of the path planning while moving from starting point to the target pose without any dynamic obstacles for both scenarios. The assigned letters in Figure

조선대학교 CHOSUN UNIVERSITY

5-20 (a), (b), and (c) are Scenario 1 results, while (d), (e), and (f) is Scenario 2 experiment results, have the following intuitions:

(a) and (d): scenarios 1 (a) and 2 (b) show the start of the navigation system in a static environment with static obstacles on Rviz and the actual environment. The pink arrow represents the target pose given to the robot by Rviz's *2D Nav Goal* tool, which is sent to the move_base node to initiate global planning.

(b), (c), (e), and (f): shows the planning operation as the robot moves towards the target goal. The position that is estimated is very close to the robot's goal. Thus, we can say that the robot repeatedly achieves a good precision to reach the goal. Table 5-5 shows the quantitative value of the target pose and the estimated pose with distance and orientation error.

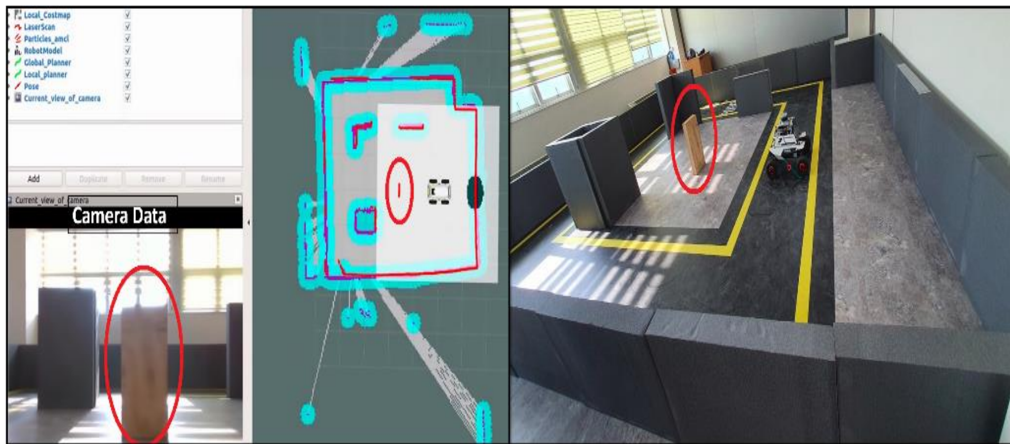
Table 5-5 Experimental comparisons of the target goal pose and the real robot estimated pose of Scenario 1 (a) and Scenario 2 (b) of Figure 5.20.

Coordinates	Robot's Starting Pose (meters)	Target Goal Pose (meters)	Robot's estimated pose after navigation (meters)
X	0.022	3.622	3.345
Y	-0.0365	-0.319	-0.222
Yaw (rad)	0.0096	2.66	2.83
Yaw (degree)	0.55	152.36	161.96
Absolute distance and orientation errors			
Absolut X-Distance Error (merers)	0.277		
Absolute Y-Distance Error (merers)	0.097		
Absolute Orientation Error (rad)	0.17		

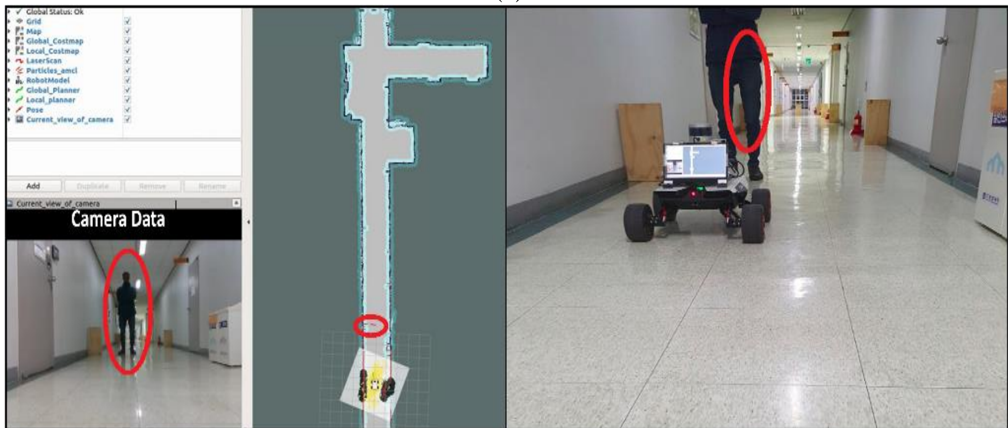
조선대학교
CHOSUN UNIVERSITY

ii. Navigation of mobile robot inside the actual environment with dynamic obstacles

We added a dynamic obstacle with a size of 30cm by 60cm wooden box in Scenario 1 and a person in Scenario 2 to test the navigation system in the actual environment with the Scout Mini mobile robot with dynamic obstacles, as shown in Figure 5-22. The dynamic obstacles in Figure 5-22 are detected by raw lidar sensor data before the navigation system is launched, as indicated by the red line on Rviz. Once the navigation begins, the local costmap is generated using the robot sensor readings to inflate the dynamic obstacles.



(a)

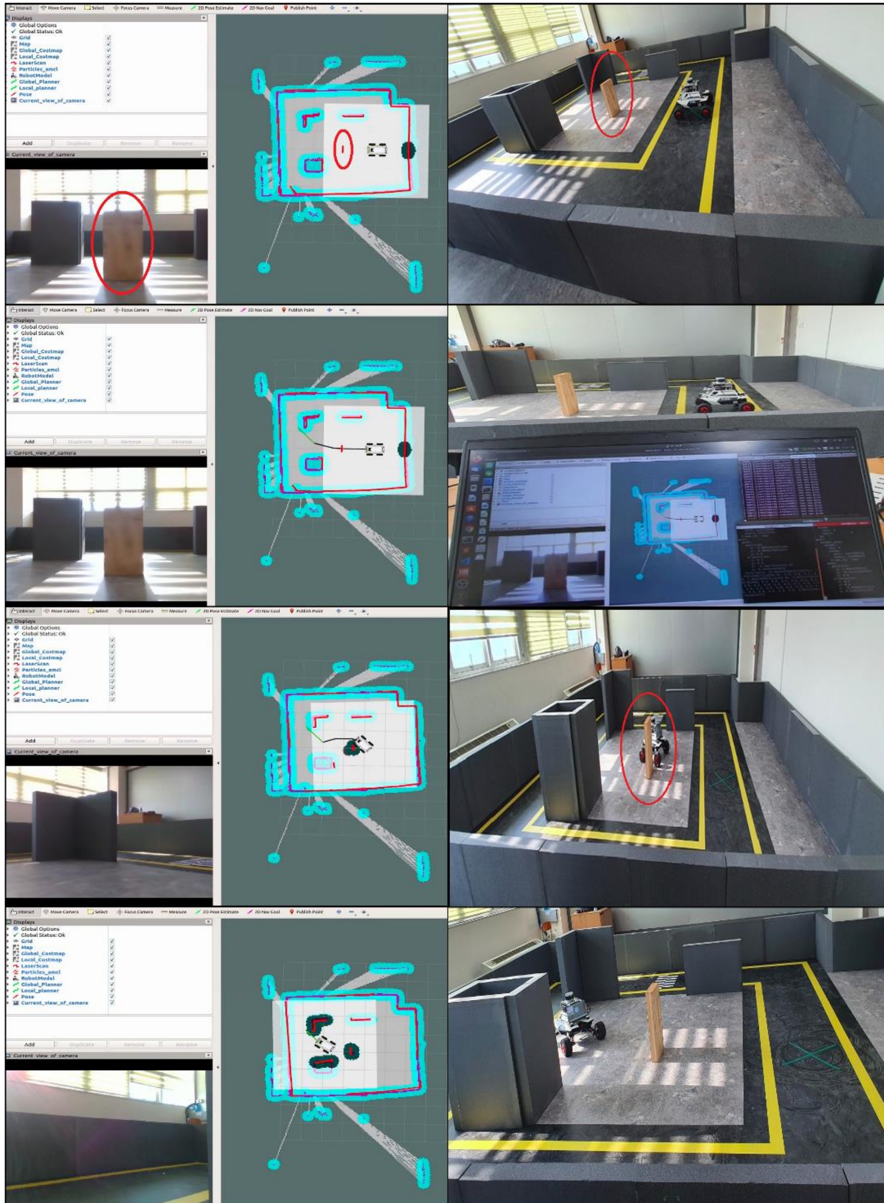


(b)

Figure 5-22 Dynamic obstacles on Rviz and the actual environment from left to right for Scenario 1 (a) and Scenario 2 (b).

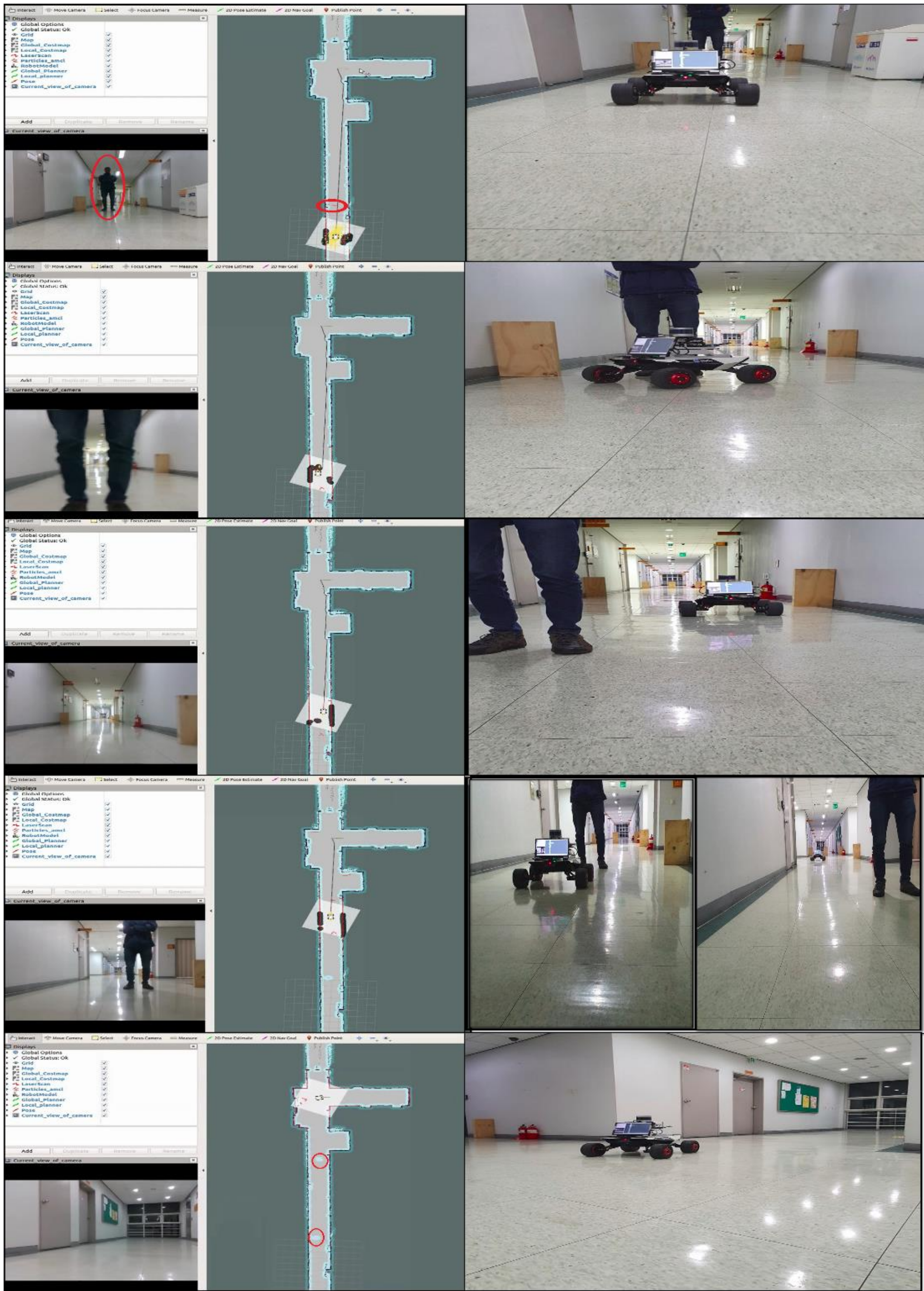
조선대학교
CHOSUN UNIVERSITY

The local planner (DWA algorithm discussed in Section 3 of the thesis) uses this local costmap to calculate the local plans shown in Figure 5-23 for both scenarios. Table 5.6 also shows the quantitative value of the target pose and the estimated pose with distance and orientation when the dynamic obstacle is placed in the robot's path.



(a)

조선대학교
CHOSUN UNIVERSITY



(b)

Figure 5-23 Autonomous navigation of a mobile robot in the actual indoor environment of the static map with dynamic obstacles for Scenario 1 and Scenario 2.

조선대학교
CHOSUN UNIVERSITY

Table 5-6 Experimental comparisons of the target goal pose and the real robot estimated pose of Scenario 1 (a) and Scenario 2 (b) for the dynamic obstacle.

Coordinates	**Robot's Starting Pose (meters)**	**Target Goal Pose (meters)**	**Robot's estimated pose after navigation (meters)**
X	0.022	3.0085	2.9046
Y	-0.0365	-0.3133	-0.3914
Yaw (rad)	0.0096	2.34	2.63
Yaw (degree)	0.55	133.976	151.138
Absolute distance and orientation errors			
Absolut X-Distance Error (merers)	0.1039		
Absolute Y-Distance Error (merers)	0.0781		
Absolute Orientation Error (rad)	0.29		

(a)

Coordinates	**Robot's Starting Pose (meters)**	**Target Goal Pose (meters)**	**Robot's estimated pose after navigation (meters)**
X	0.0	28.734	28.726
Y	0.0	0.823	1.075
Yaw (rad)	0.0	1.5708	1.99
Yaw (degree)	0.0	90.0	114.08
Absolute distance and orientation errors			
Absolut X-Distance Error (merers)	0.008		
Absolute Y-Distance Error (merers)	0.252		
Absolute Orientation Error (rad)	0.4192		

(b)

The experimental results demonstrated that the absolute distance and orientation error values differed from the simulation values due to noises in the sensors (wheel encoder, lidar),

조선대학교
CHOSUN UNIVERSITY

the slipperiness of the ceramic environment we used for the experiment, the robot wheels, and the effects of parameters in AMCL and planners discussed in section 4 of the thesis. We observed a distance error range of 3cm to 350cm and an orientation error range of 0.08rad to 0.45rad after extensive testing. Tuning parameters carefully improves the navigation system's performance. Table 5-7 shows the major parameters that we used for the navigation of the move_base node. Section 4 of this thesis discusses the meaning and functions of these parameters.

Table 5-7 Some of the navigation parameters used during the experiment of Scenario 1 and Scenario 2. (a) Global planner params, (b) Global costmap params, (c) DWA local planner params, (d) local costmap params, and (e) common costmap params

	Values
lethal_cost	253
neutral_cost	66
cost_factor	0.55

(a)

	Values
update_frequency	0.3
publish_frequency	15.0
cost_scaling_factor	5.0

(b)

	Values
xy_goal_tolerance	0.15
yaw_goal_tolerance	0.4
sim_time	3
sim_granurality	0.1
path_distance_bias	34.0
goal_distance_bias	24.0
occidist_scale	0.01
vx_sample	20
vy_sample	0
vth_sample	40

(c)

	Values
update_frequence	5.0
publish_frequency	2
static_map	false
rolling_windows	true
width	5
height	5
resolution	0.05
cost_scaling_factor	10.0

(d)

조선대학교
CHOSUN UNIVERSITY

	Values
footprint	[[-0.33, -0.33], [-0.33, 0.33], [0.33, 0.33], [0.33, -0.33]]
inflation_radius	0.3
obstacle_range	1.5
raytrace_range	3.5
transform_tolerance	0.13

5.3. Development of Graphical User Interface

The Graphical User Interface (GUI) designed to control the mobile robot's navigation in both simulation and hardware implementation. The experimental implementation of the GUI for the mobile robot will be the focus of this thesis.

The PyQt toolkit is used to create the robot's graphical user interface. PyQt is a Python binding for the Qt cross-platform widget toolkit and application framework [50]. The QT company develops Q that is used for the development of user interfaces and other applications.

QT can be installed in Ubuntu via the Advanced Packaging Tool (APT). To install, we can use the following command Qt/Qt SDK: "***$ sudo apt-get install qt-sdk***". This command installs the whole QT SDK and its libraries, which are required to complete our designed GUI. After installing the Qt SDK, PyQt must be installed on Ubuntu to bind with Qt cross-platform.

The Qt designer is used to create and insert controls into the Qt GUI. The Qt graphical user interface (GUI) is an XML file that has information about its controls and components. To control the robot with Qt GUI, we must first create the platform in Qt designer. The Qt designer provides a variety of options and tools to make the user interface simple and convenient. Figure 5.24 depicts the Qt designer platform and used *Qt 5 Designer* to develop the GUI.

조선대학교
CHOSUN UNIVERSITY

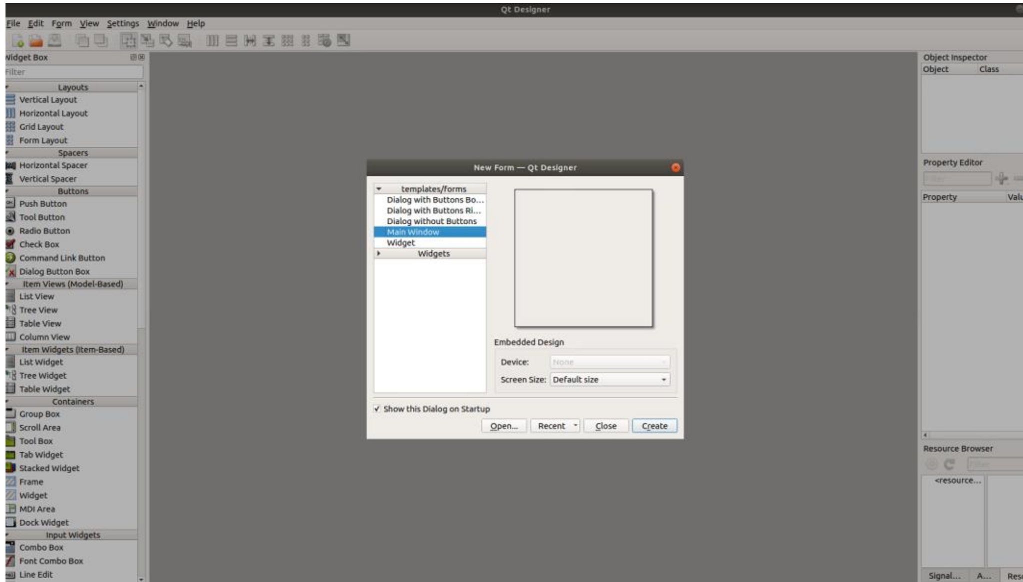


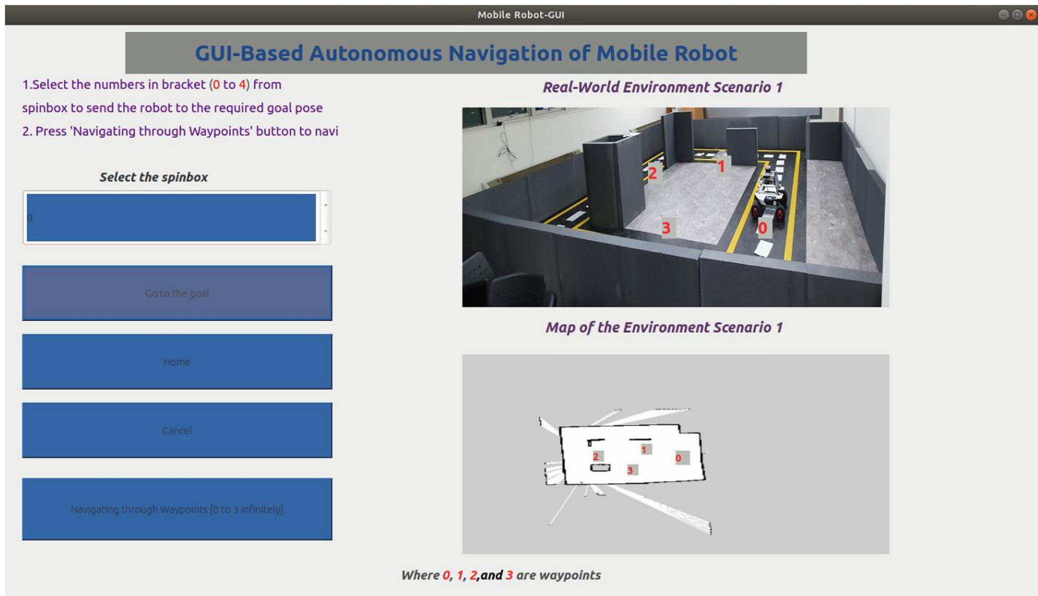
Figure 5-24 Qt 5 Designer tool.

To use the tool, we must first create an empty widget by selecting the *Widget* option from the *New Forum* window lists. The fundamental building blocks of the Qt graphical user interface are Qt widgets. Using this tool, we created an application for a mobile robot's autonomous navigation in a mapped environment via a different waypoint based on the pose that we want the robot to take, as discussed in the following section.

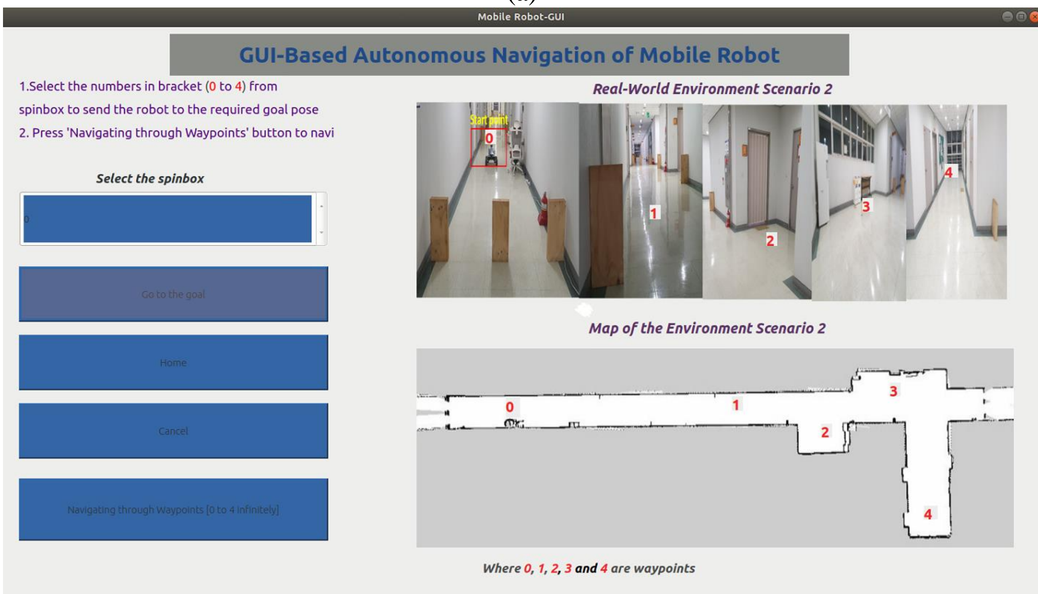
5.4. GUI-Based Autonomous Waypoint Navigation of Mobile Robot

The main goal of developing a GUI is easy to control a mobile robot to send it to the desired location and cancel the operation at any time without having to know complex commands to start and stop the robot. PyQt, ROS, and the Python interface are used to create the GUI. Figure 5-25 shows the developed graphical user interface platform for controlling the robot navigation system in both scenarios.

조선대학교
CHOSUN UNIVERSITY



(a)



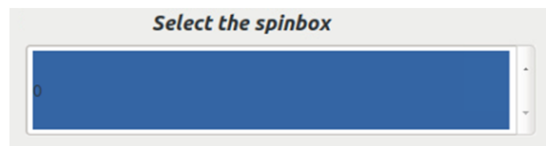
(b)

Figure 5-25 The designed GUI for autonomous navigation of a mobile robot through different waypoints(a), (b): GUI platforms that contain the actual environments and the map of the environments on the right side, and also contain 5 tools for controlling the navigation system for Scenario (a) and Scenario (b).

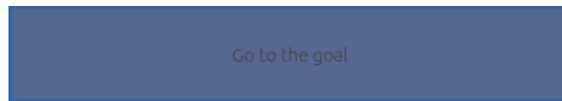
조선대학교
CHOSUN UNIVERSITY

The GUI in Figure 5-25 has the following features:

- **SpinBox:** this widget is used to insert the waypoint position numbers as shown in the environment map, ranging from 0 to 3 in Scenario 1 and 0 to 4 in Scenario 2. To navigate the robot to the target goal in Scenarios 1 and 2, we considered three pose values and four poses, respectively. We can select or insert any waypoint from the list to send the goal. It is also possible to use any waypoint we want the robot to go inside the environment. The quantitative value of these goal positions and attitudes are listed in Table 5-8.

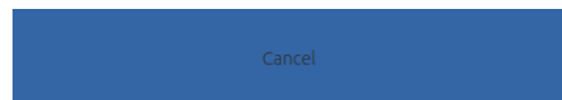


- **Go to the goal:** this button has a function called “Go**”** and it is used to command the robot to go to the goal pose to the specified position given from the spinbox. By pressing the *Go to the goal* button, the position is sent to the navigation stack, where the robot plans its path and arrives at the desired destination.



- **Home:** this button has a function called “Home” to return the robot from any location in the environment to the initial position.

- **Cancel:** this button has a "Cancel" function that allows canceling the robot's current operation. When clicking the *Cancel* button, the robot will stop moving to any point on the map.



조선대학교
CHOSUN UNIVERSITY

- **Navigate through the waypoint:** this button has a function called "navigating through waypoints" that allows the mobile robot to autonomously navigate through the poses shown in Table 5.8 in an ordered manner from 0 → 1 → 2 → 3 → 4 → 0 indefinitely.

Table 5-8 Position and attitude quantitative values of the waypoints for autonomous navigation using GUI of Figure 5.25. (a) Scenario 1 and (b) Scenario 2

Assigned numbers for waypoint poses	Absolute value of X-Position (meters)	Absolute value of Y-position (meters)	Attitude (rad)
0	0.012	0.205	3.112
1	1.348	0.819	2.542
2	2.926	0.588	0.485
3	1.125	1.332	2.086

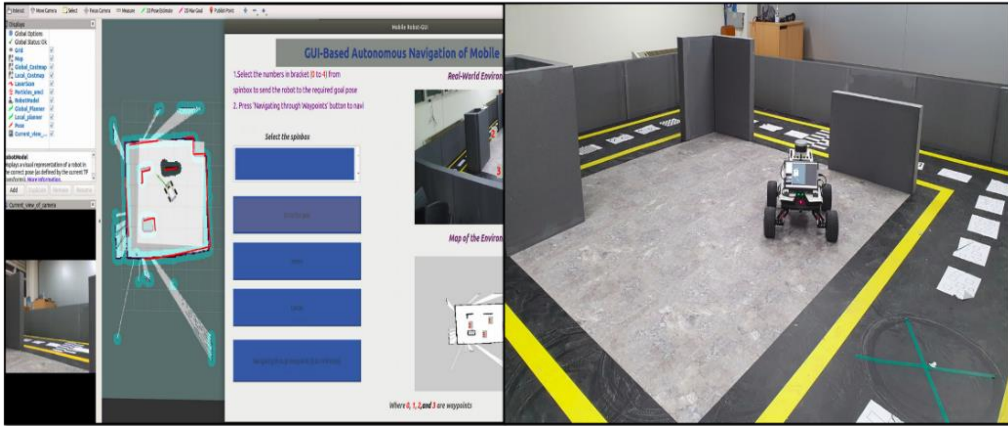
(a)

Assigned numbers for waypoint poses	Absolute value of X-Position (meters)	Absolute value of Y-position (meters)	Attitude (rad)
0	0.296	0.014	0.00046
1	14.114	0.405	2.939
2	21.516	0.679	1.012
3	27.875	2.079	2.343
4	29.519	7.939	0.105

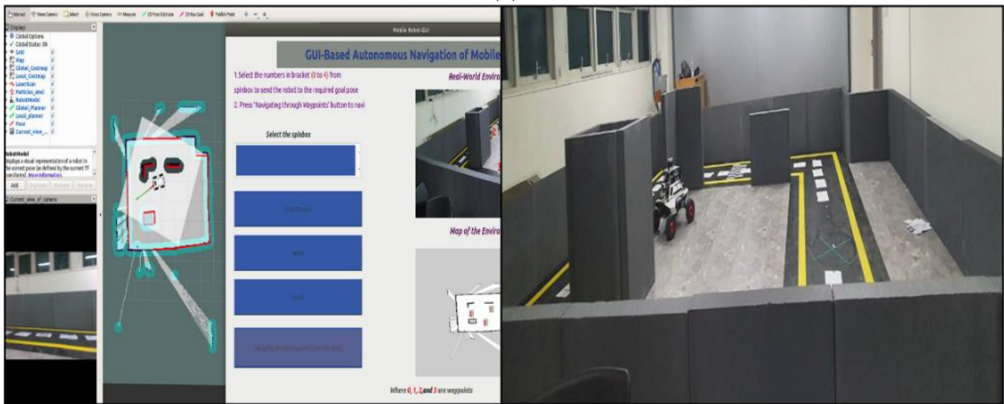
(b)

조선대학교
CHOSUN UNIVERSITY

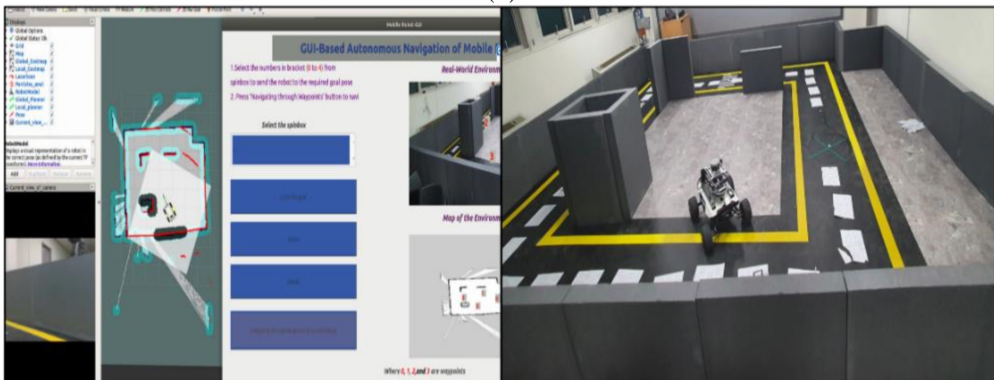
Figure 5-26 shows the real-time autonomous navigation of the Scout Mini mobile robot through all waypoints while controlled via the GUI platform in Scenarios 1.



(a)

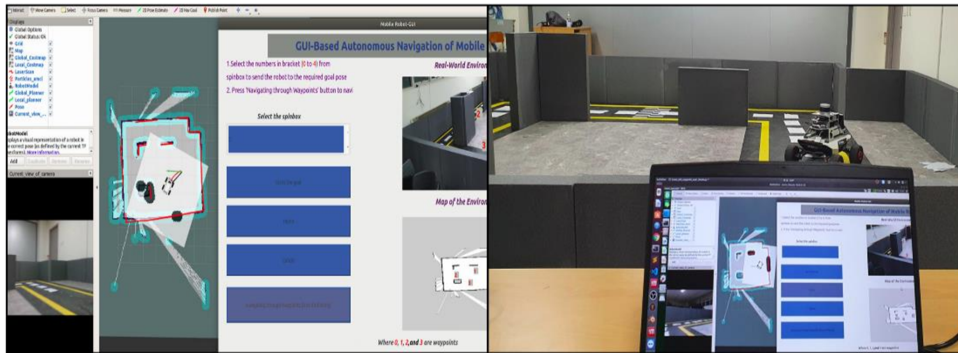


(b)



(c)

조선대학교
CHOSUN UNIVERSITY



(d)

Figure 5-26 GUI-based autonomous navigation of a mobile robot for Scenario 1. (a): shows the mobile robot's real-time autonomous navigation to the first waypoint (1) on the map using the GUI platform. (b): depicts the mobile robot's real-time autonomous navigation to the second waypoint (2) of Table 5-8 Scenario 1. (c): shows the mobile robot's real-time autonomous navigation to the third waypoint (3) on the map using the GUI platform. (d): shows the mobile robot's real-time autonomous navigation to the map's initial pose (0) using the GUI platform.

5.5. Effects of Parameters on the Mobile Robot Performance

The ROS navigation stack has many parameters that can be configured to improve the performance of a mobile robot's autonomous navigation. This section discusses some of the major parameters that influence a mobile robot's autonomous navigation. We used the Scenario-1 experiment to test the effect of each parameter on the performance of the parameters for mapping, localization, and path planning.

i. Quantitative tuning effects of Gmapping parameters

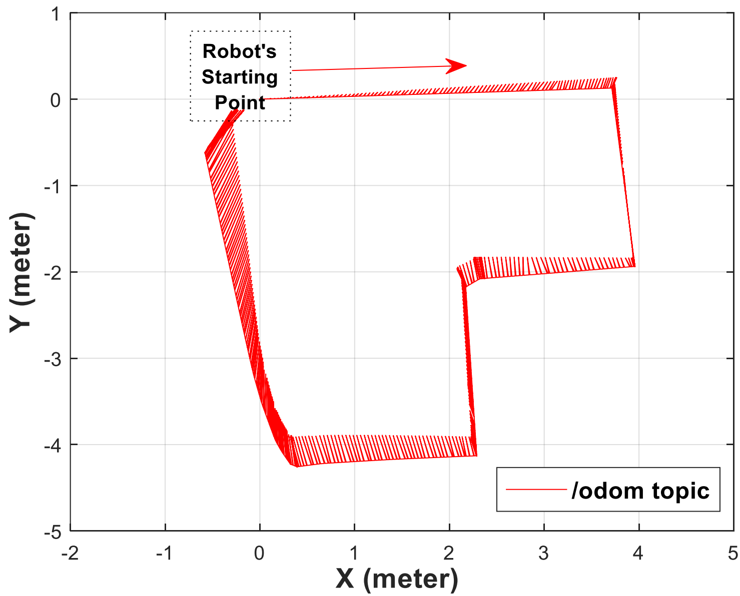
Figure 5-27 depicts the real environment (a) for testing the effect of change of parameters and the path of the robot (b) around the environment. It displays data from the robot's (encoder's) odometry with */odom* topic while teleoperating around the environment.

There are different parameters of the Gmapping algorithm such as the map's resolution (delta), the sensor's maximum range (*maxRange*), and the maximum usable range (*maxUrange*)

조선대학교 CHOSUN UNIVERSITY

of the laser parameters. From the list of Gmapping parameters, in this thesis, we will cover the effect of the map resolution parameter (delta) on the quality of the map of the environment.

(a)

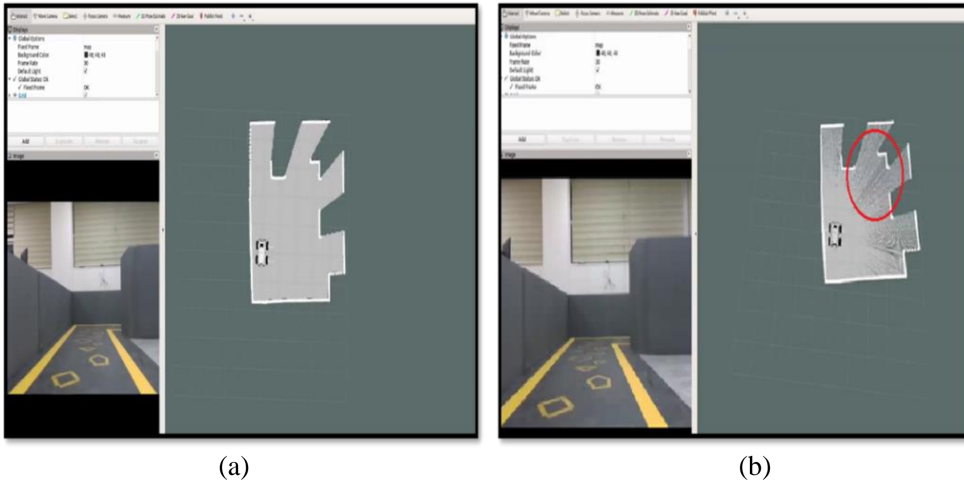


(b)

Figure 5-27 Robot path for Scenario-1environment. (a) Robot inside the environment., (b) The path of the robot with its position values.

조선대학교 CHOSUN UNIVERSITY

The *delta* parameter is defined as the environmental map's resolution expressed in meters per occupancy grid block. Figure 5-28 depicts the tuning parameter effects of delta and its effect on the time required to map the environment for two different values of delta.



(a) (b)

Figure 5-28 Effects of map resolution at the start of gmapping.

(a): the environment map with 0.05 *delta* (map resolution) values. As shown in the figure, the spaces around the environment are completely covered, and it takes less time to collect all of the environment's measurement data.

(b): In contrast, when a *delta* value of 0.01 is used, the algorithm may miss important environmental features and it takes a long time to obtain all measurement data from the environment, resulting in a poorly constructed map as shown with a red circle.

With the same robot speed of 0.2 m/sec and the same path, the Table 5-9 shows how long it took the algorithms to completely cover the environment in our case for two delta values.

Table 5-9 The effects of map resolution tuning on the time it took the robot to fully explore the environment.

Map resolution (delta) in meter per occupancy grid block	Time taken to map the environment (seconds)
0.05	104
0.01	124

조선대학교
CHOSUN UNIVERSITY

When the resolution parameter is tuned, using less resolution than the lidar resolution results in poor map quality. To properly tune the resolution parameter of Gmapping, the laser sensor resolution value must be checked. Table 5-9

ii. Quantitative effects of Adaptive Monte Carlo Localization parameter

Localization of a mobile robot is crucial to use the robot for autonomous navigation. AMCL (Adaptive Monte Carlo Localization) has over 40 parameters. In this thesis, we will focus on eight of them that can affect the robot's localization and navigation. This includes min_particles, max_particles, update_min_d, update_min_a, odom_alpha1, odom_alpha2, odom_alpha3, and odom_alpha4. To check the performance of the algorithm for different parameter values, we considered:

- The same environment for all parameters.
- The same path to follow to test the performance.
- The robot's initial pose is estimated manually on Rviz using a *2D pose estimate* tool by comparing it with the actual scenario.

To illustrate the effect on the localization of the above-mentioned parameters, we divided it into 3 cases:

Case-1: Number of particles.

These particles are represented as *min_particles* (for a minimum number of particles) and *max_particles* (for a maximum number of particles) which shows the pose estimation of the robot. In the experiment, we used 3 intervals to test the effect of the number of particles on the performance of pose estimation of the robot. Figure 5-29 and Figure 5-30 depict the position estimation quantitatively and the pose of the robot in Rviz in the Scenario-1 environment for the three-particle interval values respectively.

조선대학교
CHOSUN UNIVERSITY

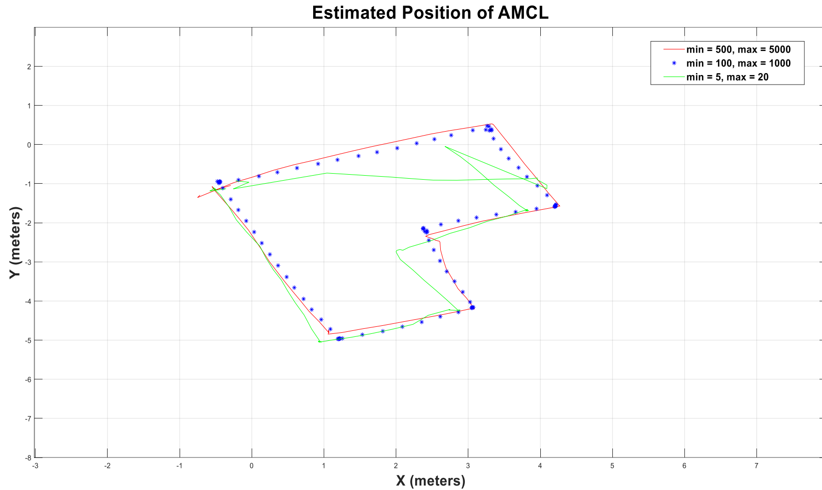
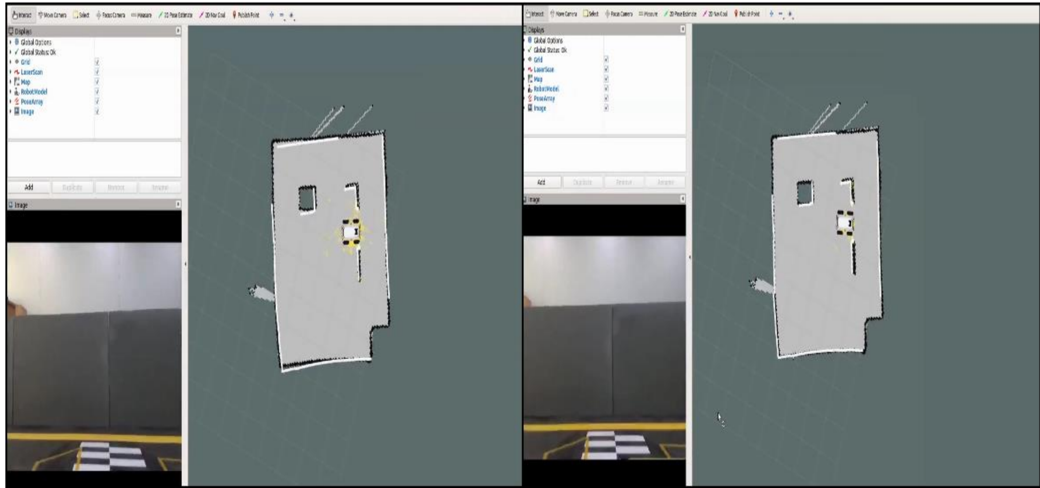


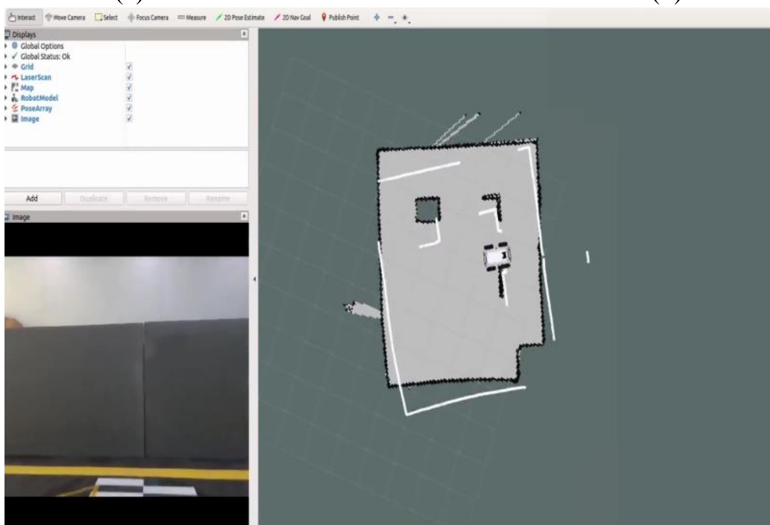
Figure 5-29 AMCL pose estimation for the three particles interval.

In the Figure 5-29, the red path represents AMCL position estimation with min_particles = 500 and max_particles = 5000. From the result of the experiment, increasing the number of particles improves localization performance, but the computation takes time. For a small indoor environment, a moderate number of particles is sufficient because the filter can obtain all of the information after collecting some data via its sensor. The blue star in Figure 5-29 represents the estimated position with min_particles = 100 and max_particles = 1000, which is quite fine due to the small area in Scenario-1, and the particles also converge to one point, as shown in Figure 5-30 (b). The third experiment is the extreme case (green path), where the min_particles = 5 and max_particles = 20. In this case, despite being teleoperated for a longer period of time to gather more information about the environment, the robot is unable to localize itself in the environment. This is due to a lack of particles during the particle filter's correction and resampling step that estimates the robot's true pose (depicted in Figure 5-30 (c)).

조선대학교
CHOSUN UNIVERSITY



(a) (b)



(c)

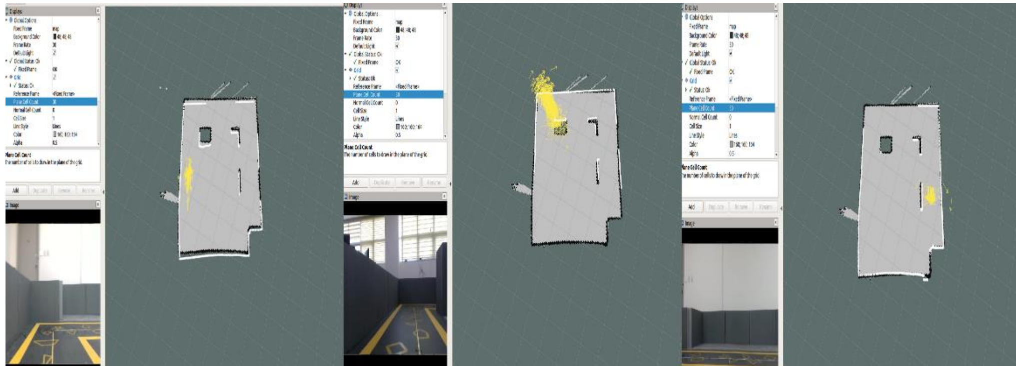
Figure 5-30 The pose estimates of the robot for the three-particle intervals on Rviz. The figures depict how the particle clouds are dispersed with the lidar matching the environment. (a) min =500, max = 5000 , (b) min = 100, max = 5000, (c) min = 5, max = 20

Case-2: The minimum translation and rotation movement before the update of a filter.

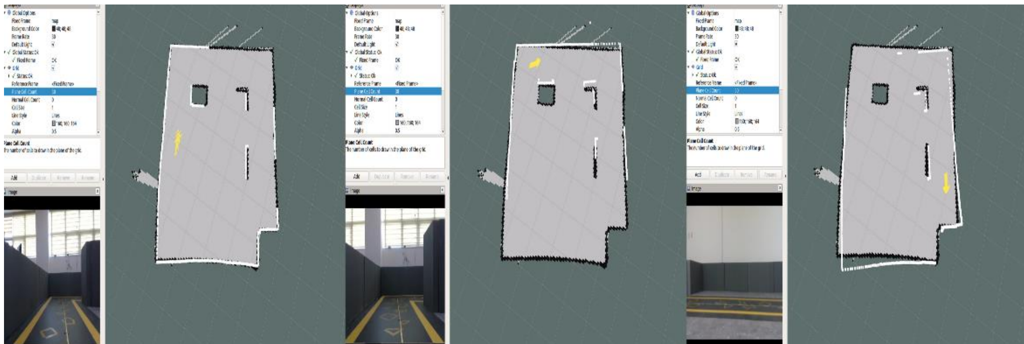
The AMCL package contains odometry information for resampling and updating the particle filters. These values are represented by the parameters update_min_d and update_min_a, which show the linear and angular motion required before performing a filter update respectively.

조선대학교
CHOSUN UNIVERSITY

To evaluate the update rate, we run the experiments with four different values. The result depicted in Figure 5-31 shows the effect of the particle point cloud dispersion. The results were obtained from the robot's various poses in the environment. We also used data from the robot's rotation around the corner to visualize the rotational motion effect in the filter update.



(a)



(b)

조선대학교
CHOSUN UNIVERSITY

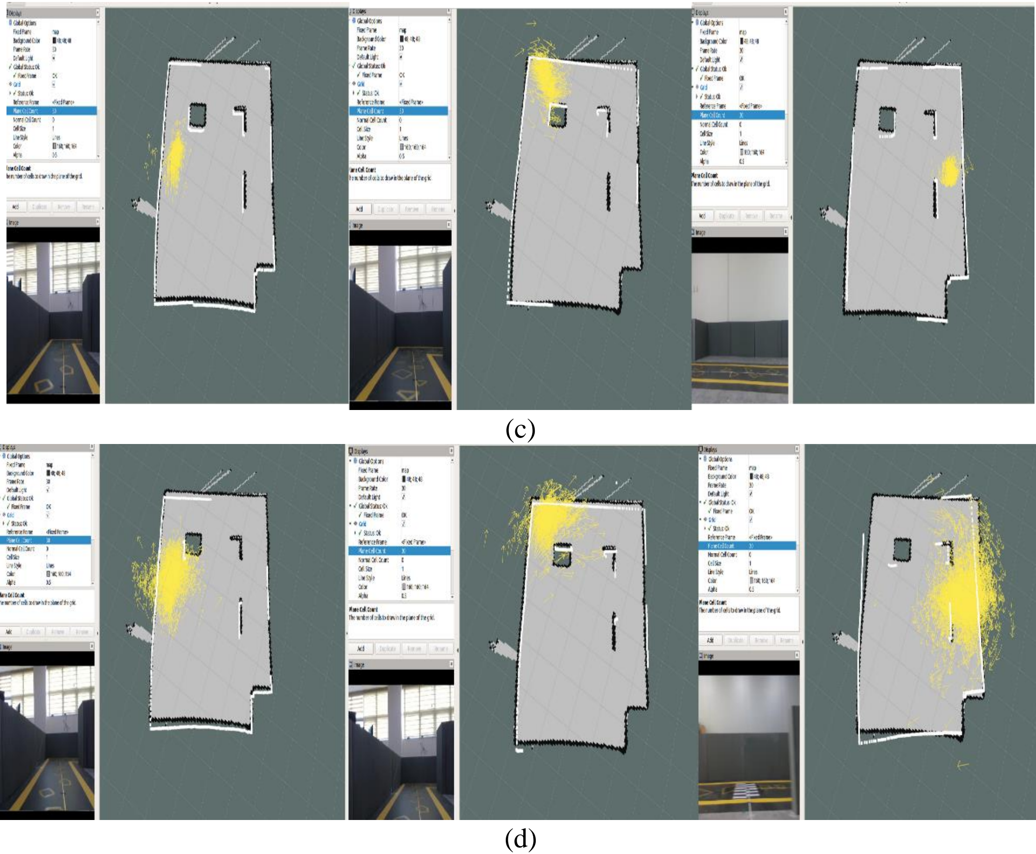


Figure 5-31 The effect of changing the value of the minimum linear and angular distance to perform filter updates. (a) update_min_d = 0.1, update_min_a = 0.1., (b) update_min_d = 0.03, update_min_a = 0.03., (c) update_min_d = 0.25, update_min_a = 0.2., (d) update_min_d = 0.75, update_min_a = 0.7.

As illustrated in Figure 5-31 (a), (b), the higher the filter update, the smaller the linear and angular distance values required before the update. This means that the particle cloud variance decreases rapidly as the robot moves a short distance. In contrast, increasing the *update_min_d* and *update_min_a* parameter values, as shown in Figure 5-32 (d), increases the particle cloud distribution while lowering the filter update as shown with the white cloud arrows results in an increase in the uncertainty of the robot pose.

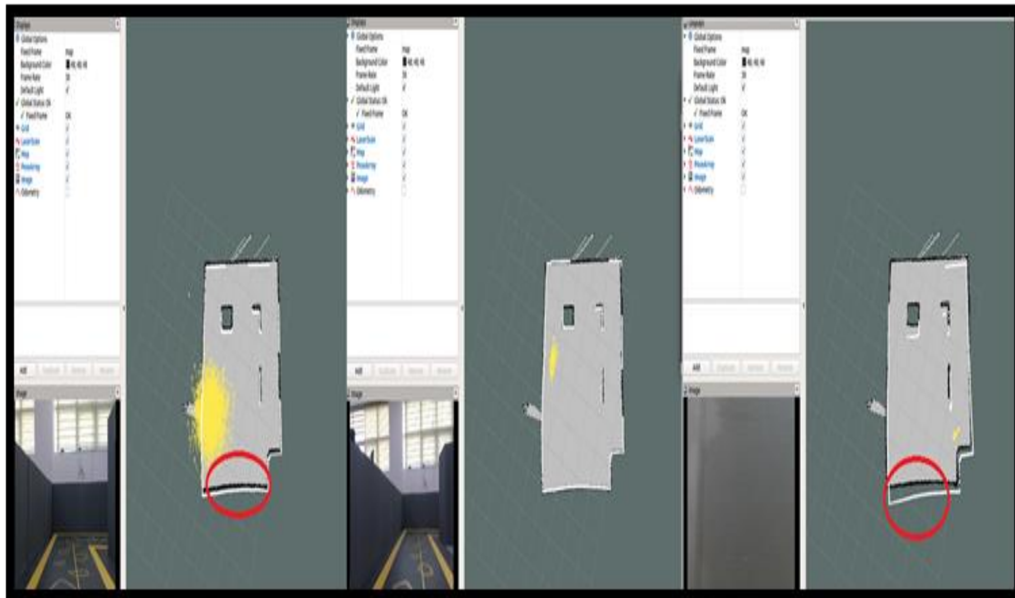
조선대학교
CHOSUN UNIVERSITY

Case-3: Odometry noise parameters.

Depending on the model of the robot, the AMCL has a total of five parameters that describe the odometry noise of the robot. This parameter is composed of the values *odom_alpha1* $(\alpha 1)$, *odom_alpha2* $(\alpha 2)$, *odom_alpha3* $(\alpha 3)$, *odom_alpha4* $(\alpha 4)$, and *odom_alpha5* $(\alpha 5)$. We do not investigate the effect of *odom_alpha5*, because it is for a robot with an *Omni* model, because the mobile robot model we used in this thesis is differential. This Omni-directional model of the robot is the one that allows a non-holonomic robot to be converted into a holonomic robot. This type of wheeled robot can move back and forth, sideways, and rotate in place. The meaning and function of the odometry noise parameters can be found in section 4 of Table 4-5.

Figure 5-32 depicts the effect of different odometry noise parameter values. The effect of odometry noise parameters is investigated and compared together with filter update parameters (update_min_d and update_min_a). The values taken for experimentation are listed in

Table 5-10.



조선대학교
CHOSUN UNIVERSITY

(a)

(b)

(c)

Figure 5-32 The effects of the odometry noise and the filter update parameters adjustments and comparisons. (a) case 1, (b) case 2, and (c) case 3

조선대학교
CHOSUN UNIVERSITY

Table 5-10 The odometry noise and filter update parameter values.

	$\alpha 1$	$\alpha 2$	$\alpha 3$	$\alpha 4$	update_min_d	update_min_a
Case1	0.05	0.05	0.08	0.08	0.1	0.1
Case2	0.05	0.05	0.08	0.08	0.65	0.5
Case3	0.55	0.55	0.65	0.65	0.25	0.2

For Figure 5.33 Case1 (a), we used the minimum values of the odometry noise parameters as well as the minimum linear and angular distance as shown in

Table 5-10. The experimental results show that the particles cloud converges quickly within six seconds because of the low odometry noise and filter update parameter values, but there is some offset or mismatch between the map of the actual environment and the laser scan data, as indicated by the red circle.

We used the same odometry noise values as Case 1 for Case 2 of Figure 5-32 but increased the values of *update_min_d* and *update_min_a* by 0.55. Even if the particles do not converge quickly, as in Case 1, the localization estimate increases and there is no offset between the map of the environment and the laser measurement data, as indicated by a green circle. The result shows that the large update_min_d and update_min_a values compensate for the small odometry noise parameter values.

In Figure 5-32 Case3, we considered the high value of the odometry noises and the moderate value of the filer updates. In this case, there is high noise in the translational as well as the rotational motion of the robot. The particles are dispersed throughout the map even if the robot gets more information from the environment when teleoperating around.

조선대학교
CHOSUN UNIVERSITY

The experimental results revealed that the odometry noise parameters selected are dependent on the quality of the wheel encoder used. Because our system has a good odometry data value in our case, using high odometry noise parameters generates a lot of uncertainty about the robot's pose, as shown in Figure 5.33 Case3. If a robot has bad odometry information, increasing the odometry noise parameters improves the robot's localization performance. The selection of the odometry noise parameters must be done carefully because it can have an impact on the robot's performance when used for autonomous navigation.

조선대학교
CHOSUN UNIVERSITY

6. CONCLUSION

In this thesis, we have done both simulation and practical experiments to test and validate the autonomous navigation algorithm of a mobile robot that is developed using the ROS platform. The lidar data is used for mapping, localization of the robot inside the environment using particle filters, path planning (global and local), and navigating through the environment from one location to the goal pose while avoiding static and dynamic obstacles. We used a 16-channel Velodyne sensor to generate the map of the two-dimensional environment and to detect obstacles in the 2D map. We also developed a Graphical User Interface (GUI) based autonomous navigation of a mobile robot through different waypoints of the actual environment that controls the robot using various buttons, as well as an emergency button to stop the robot at any time.

Furthermore, we analyzed and investigated some of the effects of the various parameters used for mobile robot mapping and localization. The proper tuning and understanding of each parameter's effect is critical for a mobile robot's map-based autonomous navigation. According to the experimental results, the mobile robot successfully reached the destination goal with a minimum localization error, and the performance of the robot's navigation is improved with careful parameter optimization.

조선대학교
CHOSUN UNIVERSITY

REFERENCES

[1] H. Wei, Z. Huang, Q. Yu, M. Liu, Y. Guan, and J. Tan, "RGMP-ROS: A real-time ROS architecture of hybrid RTOS and GPOS on multi-core processor," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014: IEEE, pp. 2482-2487.

[2] F. Vicentini *et al.*, "PIROS: Cooperative, safe and reconfigurable robotic companion for CNC pallets load/unload stations," in *Bringing Innovative Robotic Technologies from Research Labs to Industrial End-users*: Springer, 2020, pp. 57-96.

[3] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE access,* vol. 8, pp. 58443-58469, 2020.

[4] L. Roveda, "Adaptive interaction controller for compliant robot base applications," *IEEE Access,* vol. 7, pp. 6553-6561, 2018.

[5] I. Spectrum. "Care-O-bot." IEEE. https://spectrum.ieee.org/care-o-bot-4-mobile-manipulator (accessed Mar 13, 2022).

[6] H.-L. Cao *et al.*, "A collaborative homeostatic-based behavior controller for social robots in human–robot interaction experiments," *International Journal of Social Robotics,* vol. 9, no. 5, pp. 675-690, 2017.

[7] Z. Zhao and X. Chen, "Building 3D semantic maps for mobile robots using RGB-D camera," *Intelligent Service Robotics,* vol. 9, no. 4, pp. 297-309, 2016.

[8] S. Thrun, "Learning metric-topological maps for indoor mobile robot navigation," *Artificial Intelligence,* vol. 99, no. 1, pp. 21-71, 1998.

[9] I. Kostavelis and A. Gasteratos, "Semantic mapping for mobile robotics tasks: A survey," *Robotics and Autonomous Systems,* vol. 66, pp. 86-103, 2015.

[10] C. Landsiedel, V. Rieser, M. Walter, and D. Wollherr, "A review of spatial reasoning and interaction for real-world robotics," *Advanced Robotics,* vol. 31, no. 5, pp. 222-242, 2017.

[11] S. Kohlbrecher, O. Von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation," in *2011 IEEE international symposium on safety, security, and rescue robotics*, 2011: IEEE, pp. 155-160.

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

[13] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE transactions on Robotics,* vol. 23, no. 1, pp. 34-46, 2007.

[14] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai, and R. Vincent, "Efficient sparse pose adjustment for 2D mapping," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010: IEEE, pp. 22-29.

[15] A. Bartoli and P. Sturm, "Structure-from-motion using lines: Representation, triangulation, and bundle adjustment," *Computer vision and image understanding,* vol. 100, no. 3, pp. 416-441, 2005.

[16] R. Gomez-Ojeda, F.-A. Moreno, D. Zuniga-Noël, D. Scaramuzza, and J. Gonzalez-Jimenez, "PL-SLAM: A stereo SLAM system through the combination of points

조선대학교
CHOSUN UNIVERSITY

and line segments," *IEEE Transactions on Robotics,* vol. 35, no. 3, pp. 734-746, 2019.
[17] R. Guo, K. Peng, W. Fan, Y. Zhai, and Y. Liu, "RGB-D SLAM using point–plane constraints for indoor environments," *Sensors,* vol. 19, no. 12, p. 2721, 2019.
[18] X. Zhang, W. Wang, X. Qi, Z. Liao, and R. Wei, "Point-plane slam using supposed planes for indoor environments," *Sensors,* vol. 19, no. 17, p. 3795, 2019.
[19] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *2007 6th IEEE and ACM international symposium on mixed and augmented reality*, 2007: IEEE, pp. 225-234.
[20] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE transactions on robotics,* vol. 33, no. 5, pp. 1255-1262, 2017.
[21] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," *IEEE robotics & automation magazine,* vol. 13, no. 2, pp. 99-110, 2006.
[22] L. Joseph, *Robot Operating System (ROS) for Absolute Beginners*. Springer, 2018.
[23] M. Quigley, B. Gerkey, and W. D. Smart, *Programming Robots with ROS: a practical introduction to the Robot Operating System*. " O'Reilly Media, Inc.", 2015.
[24] A. M. R. "ROS/concepts." ROS wiki. http://wiki.ros.org/ROS/Concepts (accessed Apr. 10, 2022).
[25] I. Saito. "Packages." ROS wiki. http://wiki.ros.org/Packages (accessed Apr. 12, 2022).
[26] M. Yasuyuki. "ROS Distributions." ROS wiki. http://wiki.ros.org/Distributions (accessed Apr. 13, 2022).
[27] Playfish. "URDF." ROS wiki. http://wiki.ros.org/urdf (accessed Apr. 13, 2022).
[28] T. Foote. "Coordinate Frames, Transforms, and TF." ROS wiki. http://wiki.ros.org/tf/Overview/Transformations (accessed Apr. 13, 2022).
[29] O. S. R. Foundation. "What is Gazebo?" Gazebo. https://gazebosim.org/tutorials?cat=guided_b&tut=guided_b1 (accessed May. 13, 2022).
[30] H. Do Quang *et al.*, "Mapping and navigation with four-wheeled omnidirectional mobile robot based on robot operating system," in *2019 International Conference on Mechatronics, Robotics and Systems Engineering (MoRSE)*, 2019: IEEE, pp. 54-59.
[31] M. Iovino, "Navigation and Grasping with a Mobile Manipulator: from Simulation to Experimental Results," 2019.
[32] W. B. Sebastian Thrun, Dieter Fox, *Probabilistic Robotics*. ©Massachusetts Institute of Technology, 2006.
[33] S. A. Fadzli, S. I. Abdulkadir, M. Makhtar, and A. A. Jamal, "Robotic indoor path planning using dijkstra's algorithm with multi-layer dictionaries," in *2015 2nd International Conference on Information Science and Security (ICISS)*, 2015: IEEE, pp. 1-4.
[34] D. Ferguson, M. Likhachev, and A. Stentz, "A guide to heuristic-based path planning," in *Proceedings of the international workshop on planning under uncertainty for autonomous systems, international conference on automated planning and scheduling (ICAPS)*, 2005, pp. 9-18.

조선대학교
CHOSUN UNIVERSITY

[35] S. I. Gass and C. M. Harris, "Encyclopedia of operations research and management science," *Journal of the Operational Research Society,* vol. 48, no. 7, pp. 759-760, 1997.

[36] P. Marin-Plaza, A. Hussein, D. Martin, and A. d. l. Escalera, "Global and local path planning study in a ROS-based research platform for autonomous vehicles," *Journal of Advanced Transportation,* vol. 2018, 2018.

[37] 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, 1997.

[38] E. P. Eitan Marder-Eppstein. "base_local_planner." ROS Wiki. http://wiki.ros.org/base_local_planner (accessed May 10, 2022).

[39] E. P. Eitan Marder-Eppstein. "dwa_local_planner." ROS Wiki. http://wiki.ros.org/dwa_local_planner (accessed May 10, 2022).

[40] R. Tang, X. Q. Chen, M. Hayes, and I. Palmer, "Development of a navigation system for semi-autonomous operation of wheelchairs," in *Proceedings of 2012 IEEE/ASME 8th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications*, 2012: IEEE, pp. 257-262.

[41] N. Kwak, I.-K. Kim, H.-C. Lee, and B.-H. Lee, "Analysis of resampling process for the particle depletion problem in FastSLAM," in *RO-MAN 2007-The 16th IEEE International Symposium on Robot and Human Interactive Communication*, 2007: IEEE, pp. 200-205.

[42] B. Gerkey. "gmapping." ROS Wiki. http://wiki.ros.org/gmapping (accessed May 11, 2022).

[43] S. Thrun, "Particle Filters in Robotics," in *UAI*, 2002, vol. 2: Citeseer, pp. 511-518.

[44] B. P. Gerkey. "amcl." ROS Wiki. http://wiki.ros.org/amcl (accessed May 12, 2022).

[45] E. P. Eitan Marder-Eppstein. "move_base." ROS Wiki. http://wiki.ros.org/move_base (accessed May 12, 2022).

[46] K. Zheng, "Ros navigation tuning guide," in *Robot Operating System (ROS)*: Springer, 2021, pp. 197-226.

[47] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," in *Proceedings 1999 ieee international conference on robotics and automation (Cat. No. 99CH36288C)*, 1999, vol. 1: IEEE, pp. 341-346.

[48] E. M.-E. Kurt Konolige. "navfn." ROS Wiki. http://wiki.ros.org/navfn (accessed May 13, 2022).

[49] D. Lu. "global_planner." ROS Wiki. http://wiki.ros.org/global_planner (accessed May 13, 2022).

[50] J. M. Willman, "Getting Started with PyQt," in *Beginning PyQt*: Springer, 2022, pp. 1-11.

조선대학교
CHOSUN UNIVERSITY

PUBLICATIONS

Journal

1. *Yebasse, M., Shimelis, B., Warku, H., Ko, J. and Cheoi, K.J., 2021. Coffee Disease Visualization and Classification. Plants, 10(6), p.1257.*

Conferences

1. *Warku, H. T., Ko, N. Y., Yeom, H. G., & Choi, W. (2021, October). Three-Dimensional Mapping of Indoor and Outdoor Environment Using LIO-SAM. In 2021 21st International Conference on Control, Automation and Systems (ICCAS) (pp. 1455-1458). IEEE.*
2. *Warku, Henok Tegegn, and Nak Yong Ko*, et al. "Indoor Environment Mapping Based on ROS Using GMapping Algorithm and Lidar Sensor." 제어로봇시스템학회 국내학술대회 논문집 (2021): 401-402.*
3. *Warku, Henok Tegegn, Nak Yong Ko*, Gyeonsub Song, and Da Bin Jeong. "Pose Estimation of a Mobile Robot in a Small Indoor Workspace Using Ultrasonic Beacons."*

조선대학교
CHOSUN UNIVERSITY

ACKNOWLEDGEMENTS

First, I would like to thank the almighty God for letting me through all the difficulties. I thank him for giving me the strength and ability to learn, understand and complete this research.

Second, I want to express my deepest gratitude to my esteemed advisor, Professor Nak Yong Ko (Ph.D.), for his sincerity and encouragement throughout my master's degree. His immense knowledge and plentiful support have guided me in my academic research and daily life. I am thankful for the extraordinary experiences he arranged for me and for providing opportunities for me to grow professionally. This thesis would not have been possible without his guidance from the initial step, and it is an honor to learn from him.

Besides my advisor, I would like to thank my dissertation evaluation committee members, Prof. Hong Gi Yeom and Prof. Sung Hyun You, for their valuable comments and suggestions on my research. I would like to extend my sincere thanks to my friends and Lab mates -Gyeongsub Song, Da Bin Jeong, Boeun Lee, Eyasu Derbew Tegen, and Seo Hyun Kim, for their valuable support, inspiration, and assistance throughout my studies.

Finally, my profound appreciation goes out to my beloved family and friends who directly or indirectly helped me with their prayers, encouragement, and emotional support throughout my years of study. This accomplishment would not have been possible without them. Thank you.