SEAGATE ROBOT MK. II (SIMULATION AND MIGRATION)

BY

ARIN MADNURAK

THANAPAT PAKORNKITTIBOWORN

THANAWIT NORKAM

THITISART THITATHAN

A PROJECT SUBMITTED IN PARTIAL FULIFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF BACHELOR OF ENGINEERING IN ROBOTICS AND AI

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ACADEMIC YEAR 2022

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content, and cite the document when use.

FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

PROJECT CERTIFICATE

Project Title	Seagate Robot Mk.II (Simulation and Migration)	
Student Name	Mr. Arin Madnurak	Student ID: 62011093
	Mr. Thanapat Pakornkittiboworn	Student ID: 62011273
	Mr. Thanawit Norkam	Student ID: 62011275
	Mr. Thitisart Thitathan	Student ID: 62011280
Degree	Bachelor of Engineering in Robotics and AI	

Project Advisor Signed: Dr. Poom K.

(Dr. Poom Konghuayrob)

Project Co-Advisor Signed: Sarucha Y.

(Mr. Sarucha Yanyong)

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Project Title	SEAGATE Robot Mk. II (Simulation and Migration)	
Student Name	Mr. Arin Madnurak	Student ID: 62011093
	Mr. Thanapat Pakornkittiboworn	Student ID: 62011273
	Mr. Thanawit Norkam	Student ID: 62011275
	Mr. Thitisart Thitathan	Student ID: 62011280
Degree	Bachelor of Engineering in Robotics and AI	
Project Advisor	Dr. Poom Konghuayrob	
Project Co-Advisor	Mr. Sarucha Yanyong	
Academic Years	2022	

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

ABSTRACT

Open-source software plays a pivotal role in the advancement of technology, particularly in the field of robotics. The continuous evolution of open-source software follows a distinct life cycle, comprising various stages such as development, testing, deployment, and maintenance. One such prominent open-source software platform is the Robot Operating System (ROS), which has revolutionized the development of robotics software.

This report aims to explore the life cycle of open-source software, with a specific focus on the two major versions of ROS: ROS1 and ROS2. By understanding the nuances of these versions, educators in the field of mobile robotics can effectively adapt their teaching methodologies to incorporate the latest advancements in software development. Additionally, the report delves into the topic of robot simulation in Gazebo, a powerful simulation environment commonly used in conjunction with ROS.

By comprehending the open-source software life cycle, the differences between ROS1 and ROS2, and the utilization of Gazebo for robot simulation with 3D maps, educators can tailor their teaching approaches to equip students with the necessary skills and knowledge required in the dynamic field of mobile robotics. This report serves as a valuable resource for educators seeking to stay updated with the latest advancements in open-source software and effectively impart this knowledge to their students.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

ACKNOWLEDGEMENTS

We would like to take this opportunity to express our utmost gratitude to our beloved advisor, Dr. Poom Konghuayrob, for their invaluable guidance, support, and mentorship throughout the duration of our project. Under their esteemed supervision, we were able to apply the knowledge and skills acquired during our 4-year study at KMITL (King Mongkut's Institute of Technology Ladkrabang) in a practical and meaningful manner.

Dr. Poom Konghuayrob, your unwavering commitment to our success has been a constant source of inspiration. Your expertise in the field and dedication to teaching have not only enhanced our technical understanding but also broadened our horizons. Your open-door policy and willingness to provide guidance and constructive feedback have empowered us to overcome challenges and achieve our goals. Dr. Poom's guidance was instrumental in shaping the trajectory of our project, the SEAGATE Robot Mk. II. Your keen insights and ability to think critically enabled us to navigate complex problems and develop innovative solutions. Your encouragement to explore new avenues and push the boundaries of our capabilities has fostered a spirit of curiosity and intellectual growth within us.

In addition, we would like to extend our heartfelt appreciation to our co-advisor, Mr. Sarucha Yanyong, for his effort in supporting us and providing applicable feedback throughout the course of our project. Mr. Sarucha Yanyong's dedication to our development, and willingness to invest your time and expertise have been truly commendable. Your attention to detail and commitment to excellence have helped refine our work and ensure its quality.

As we conclude this report, we are filled with a profound sense of gratitude for the opportunity to work under the guidance and mentorship of Dr. Poom Konghuayrob and Mr. Sarucha Yanyong. Their wisdom, expertise, and unwavering support have left an indelible mark on our lives. We are confident that the knowledge and experiences we have gained under their tutelage will serve as a solid foundation as we embark on the next chapter of our journey.

Thank you.

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content, and cite the document when use.

TABLE OF CONTENT

PROJECT CERTIFICATE .... **II**
ABSTRACT .... **IV**
ACKNOWLEDGEMENTS .... **V**
TABLE OF CONTENTS .... **VI**
LIST OF FIGURES .... **IX**
LIST OF TABLES .... **X**
CHAPTER 1: INTRODUCTION **1**
1.1 Background and Significance 1
1.1.1 Algorithm Development 1
1.1.2 System Testing 1
1.1.3 Educational Material for Junior Researchers 2
1.1.4 The Ubuntu Life Cycle 2
1.1.5 The Upcoming ROS2 2
1.1.6 Future Research Compatibility 2
1.2 Research objectives 2
1.2.1 Algorithm Optimization 3
1.2.2 System Validation 3
1.2.3 Educational Material Development 3
1.2.4 Further Software Compatibility 3
1.2.5 ROS2 Package Education 3
1.2.6 Further Development Compatibility 4
CHAPTER 2: CONCEPT, THEORIES AND RELATED RESEARCH **5**
2.1 Theory 5
2.1.1 Physics-based simulation 5
2.1.2 Sensor simulation 5
2.1.3 Plug-in architecture 5
2.1.4 Real-time performance 5

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

2.2 Practical Theory 5
2.2.1 Mobile Robot 5
2.2.2 SEAGATE Robot 6
2.2.3 Linux Operating System 7
2.2.4 Ubuntu 7
2.2.5 Robot Operating System 7
2.2.6 Gazebo integration with ROS2 7
2.2.7 3D Modelling 8
2.2.8 3D Format for Gazebo 9
2.2.9 Robot and Environment Modelling 11
2.2.10 Mobile Robot Control 12
2.2.11 ROS package migration 17
2.2.12 ROS1 VS ROS2 17
2.2.13 ROS OS Support 23
2.2.14 ROS Package Components 23
2.3 Related Research 24
CHAPTER 3: METHODOLOGY 25
3.1 Introduction 25
3.2 Simulation 25
3.2.1 Specify the Simulation's environment 26
3.2.2 Blueprint and Measurement for 3D creations 30
3.2.3 Create 3D map 31
3.2.4 Seagate robot for simulation 33
3.3 Migration 51
3.3.1 CMakeLists.txt 51
3.3.2 Package.xml 53
3.3.3 Main source code 54
3.3.4 Launch file (.xml) 57
CHAPTER 4: EXPERIMENTAL RESULT 58

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

4.1 Simulation 58
4.1.1 Navigation and path planning 59
4.1.2 Robot sensing 59
4.1.3 Teleoperation capability 60
4.1.4 3D maps benefit 61
4.2 Migration 63
4.2.1 Topic Monitoring 63
4.2.2 Visualization 64
4.2.3 SLAM 65
CHAPTER 5: CONCLUSION **67**
REFERENCES **68**
BIOGRAPHY **70**

King Mongkut's Institute of Technology Ladkrabang
This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

LIST OF FIGURES

Figure 2. 1 Seagate Robot 6
Figure 3. 1 Seagate Robot MK II Simulation 25
Figure 3. 2 HM Robotics Laboratory..27
Figure 3. 3 HM Building 6th floor 28
Figure 3. 4 E12 Building 12th floor 29
Figure 3. 5 Laboratory & Building floor blueprints 30
Figure 3. 6 Measurement 31
Figure 3. 7 HM Robotics Laboratory 3D map 32
Figure 3. 8 3D robot simulation working flow 33
Figure 3. 9 3D model of Seagate mobile robot...34
Figure 3. 10 Seagate mobile robot measurement...35
Figure 3. 11 Seagate model with visual and collision representation 36
Figure 3. 12 URDF Structure of Seagate mobile robot 37
Figure 3. 13 Seagate mobile robot's TF2 frame 38
Figure 3. 14 Visual representation of wheel_left..39
Figure 3. 15 collision representation of wheel_left 40
Figure 3. 16 Communication with other nodes during run a simulation 43
Figure 3. 17 Seagate mobile robot laser scan range 48
Figure 4. 1 Seagate mobile robot in simulation..58
Figure 4. 2 Seagate mobile robot simulation running navigation NAV2 59
Figure 4. 3 Visualize the output topic by rviz ,rqt 59
Figure 4. 4 Seagate mobile robot running teleop_key 60
Figure 4. 5 Final of 3D HM Robotics Laboratory map 61
Figure 4. 6 Final view of 3D HM 6th floor map...62
Figure 4. 7 Final view of E12 12th floor map 62
Figure 4. 8 ROS1 rqt_graph...63
Figure 4. 9 ROS2 rqt_graph...64
Figure 4. 10 Visualization via ROS2 RVIZ...65
Figure 4. 11 SLAM via cartographer..66

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

LIST OF TABLES

Table 2. 1 Weight parameters of “SEAGATE mobile robot” 6

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

CHAPTER 1
INTRODUCTION

1.1 Background and significance

Seagate is an innovative mobile robot designed for various applications such as warehouse automation, coordination, and indoor navigation. To ensure its efficient operation and performance, Algorithm test and validation is required before real-world deployment. Mobile robot simulation provides a valuable platform to accomplish this by creating virtual environments that closely mimic the robot's behavior and interactions with its surroundings. Furthermore, the Seagate robot's packages will be deprecated. The current framework used in Seagate robot is still ROS1 and Ubuntu 18.04 LTS. The End-of-Life state of any open-source software is inevitable. The End-of-Life period of Ubuntu 18.04 LTS is in April 2023. The result of End-of-Life software is no further development. The package migration is a crucial part for the Seagate mobile robot. With the upcoming trend of implementing ROS2 in robotics development project and the newly introduced features, Migration allows the developer to use the old packages but changing the structure of the source code instead of creating a new package for similar tasks. The background and significance of the Seagate robot MK. II listed as follows:

1.1.1 Algorithm Development: Simulation allows for rigorous testing and refinement of Seagate's algorithms, including navigation, obstacle avoidance, localization, and mapping. By accurately simulating the robot's dynamics, sensors, and environment, researchers can experiment with different algorithms, fine-tune parameters, and evaluate their effectiveness in diverse scenarios. The iterative process leads to improved algorithm performance and robustness.

1.1.2 System Testing: Simulating Seagate's behavior and performance in various simulated environments helps identify potential issues and optimize its overall system reliability. System testing enables researchers to evaluate how the robot interacts with its environment, manages dynamic obstacles, adapts to changing conditions, and validates the correctness of

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

implemented functionalities. Simulation-based testing reduces the risks and costs associated with real-world trials.

1.1.3 **Educational Material for Junior Researchers**: The Seagate mobile robot simulation serves as an educational tool for junior researchers and students. SEAGATE mobile robot provides an accessible and safe environment for learning and experimenting with robotics concepts, algorithms, and system design principles through interactive simulations, junior researchers can gain firsthand experience, explore different scenarios, and develop a deeper understanding of mobile robot behaviors.

1.1.4 **The Ubuntu Life Cycle**: One concern about the Ubuntu life cycle as an open-source operating system is that while LTS (Long Term Support) releases are published every two years and are supported for up to 10 years, interim releases are only supported for 9 months. This means that users who choose to use interim releases need to update their systems more frequently to continue receiving support and security updates.

1.1.5 **The Upcoming ROS2:** ROS2 was developed from scratch as a completely new ROS to address the limitations of ROS1 and to make ROS compatible with industrial applications. The improvements in ROS2 over ROS1 include real-time capabilities, safety, certification, and security.

1.1.6 **Future Research Compatibility**: Future trend for Robotics development is to use ROS2. The deprecation of ROS1 prevents developers from using the existing packages for further research purposes. In addition, ROS2 also provides new features which can be helpful tools for easier robotics development.

1.2 Research objectives

The project Seagate Robot Mk. II has the goal of implementing the elements and the concept of ROS2 as a study topic for educators who are interested in the field of mobile robotics. The project also has the purpose of a demonstration for the mobile robotics project developing process. The simulation

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content, and cite the document when use.

provides a brief image of the functionality of mobile robots while the migration adapts the existing robot to the new framework of ROS2. The simulation, in addition, demonstrates the process of generating a virtual 3D map based on the pure measurement and the blueprint of a building. The research objectives in the Seagate robot MK. II can be divided in elements as follows:

1.2.1 **Algorithm Optimization**: Optimize Seagate's algorithms for navigation, path planning, obstacle avoidance, and localization through iterative simulations. Fine-tune parameters and evaluate algorithmic performance under various environmental conditions to enhance the robot's efficiency, reliability, and adaptability.

1.2.2 **System Validation**: Validate Seagate's system functionalities and performance in simulated environments. Evaluate the ability to handle complex scenarios, dynamic obstacles, sensor noise, and challenging terrains. Identify and rectify potential issues to improve the robot's overall performance and robustness.

1.2.3 **Educational Material Development**: Create educational materials, tutorials, and interactive simulations to facilitate the learning and understanding of mobile robotics concepts, particularly aimed at junior researchers and students. These resources should cover fundamental principles, algorithm implementation, and system design considerations using Seagate as a case study.

1.2.4 **Further Software Compatibility**: The deprecation of Ubuntu 18.04 LTS and ROS1 prevents development in the long-term period. Migration is the answer to the previous issue given that developers are not required to create a new package for every release of Ubuntu. Developers do not need to build a new robot or change the hardware or components in the robot given that the hardware is no longer supported.

1.2.5 **ROS2 Package Education**: The source code of ROS2 has a distinctive structure compared to ROS1. The study in ROS2 code structure is required to understand the code and consider the replacement for the old, deprecated library. The comprehension of ROS2 code structure is a

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content, and cite the document when use.

necessary knowledge and will be used more often in the migration processes.

1.2.6 Further Development Compatibility: Every release of ROS2 distro has new helpful features with ease of development. Using ROS2 in further robotics development also has the benefits of introducing and meeting specific product requirements spanning design, development, and project governance when a robot moves beyond the laboratory and into commercial use. ROS2 aims to expand the universe of applications it can support by adding enterprise features such as security and real-time communication.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

CHAPTER 2

CONCEPT, THEORIES AND RELATED RESEARCH

2.1 Theory

2.1.1 Physics-based simulation: Gazebo uses a physics engine to simulate the interactions between objects in the virtual environment, including dynamics such as gravity, friction, and collisions. Physics-based simulation allows for realistic simulations of robots and their environments.

2.1.2 Sensor simulation: Gazebo allows for the simulation of various sensors (such as cameras, lidars, and IMUs) which can be used to provide sensor data to a robot's control system. Sensor simulation allows for the testing of sensor-based navigation and perception algorithms.

2.1.3 Plug-in architecture: Gazebo uses a plug-in architecture, which allows developers to easily add new functionality and models to the simulation. Plug-in architecture includes the ability to interface with other simulations tools, such as ROS (Robot Operating System) and MATLAB.

2.1.4 Real-time performance: Gazebo uses a real-time physics engine, allowing for real-time interaction during the simulation and enabling the testing of control algorithms with high frequency.

Institute of Technology

2.2 Practical Theory

2.2.1 Mobile Robot: A mobile robot [1] is a specific type of robot or machine controlled by software utilizing sensors or other technology (i.e., camera) to identify the covered surroundings and move around the environment. Mobile robots function using a combination of artificial intelligence (AI) and physical robotic elements, such as wheels, tracks, and legs. Mobile robots are becoming increasingly popular across different business sectors. Mobile robots are also used to help with work processes and even accomplish impossible or harmful tasks for humans.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

2.2.2 SEAGATE Robot: The Seagate robot is a project of KMITL (King Mongkut s Institute of Technology Ladkrabang) senior Mr. Nattapat Koomklang. The robot is a collaboration project between the Seagate Company and KMITL. The project's objective is to create a mobile robot suitable for operating in the industrial cleanroom. The industrial cleanroom [2] is a specially engineered and carefully designed enclosed area within a manufacturing or research facility. Clean rooms allow for precise control, monitoring, and maintenance of an internal environment.

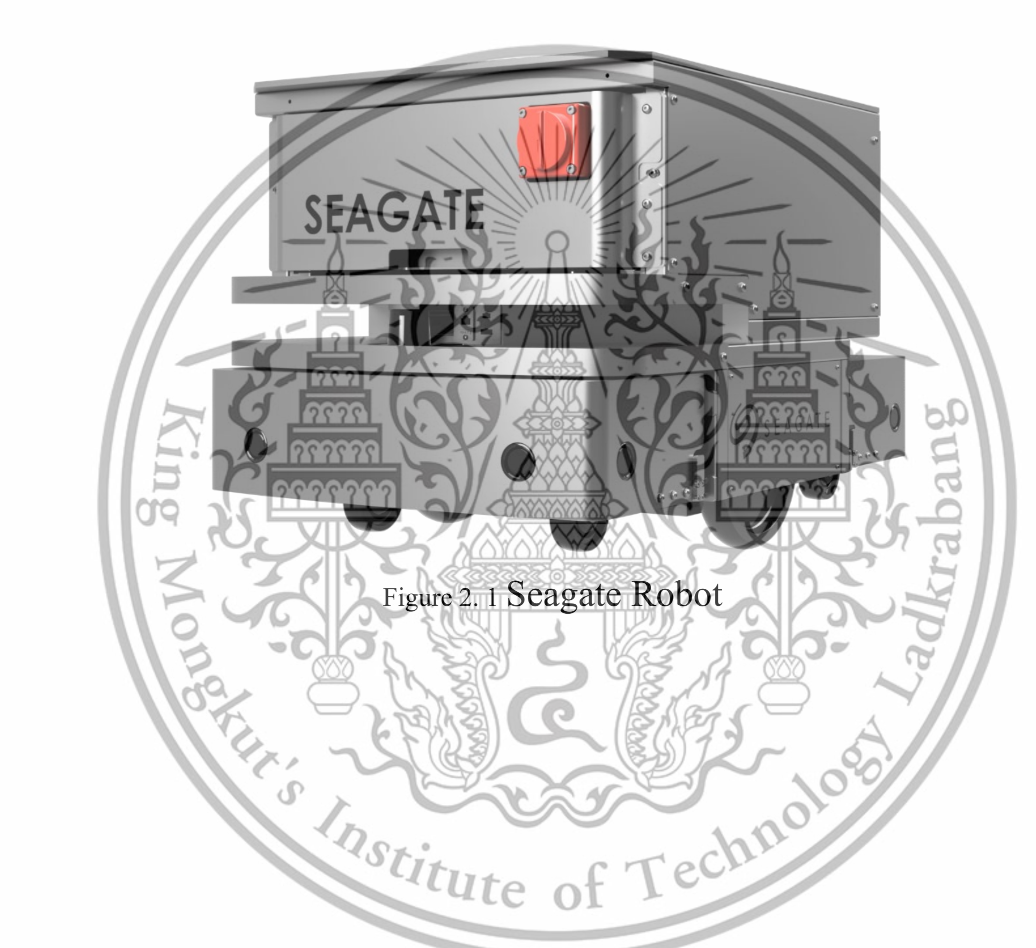


Figure 2. 1 Seagate Robot

Equipment	Weight parameter
Robot chassis with two main wheels, two brushless DC motors with break	-
On-board control, motor driver and power system	-
Li battery	-
Total	80kg

Table 2. 1 Weight parameters of "SEAGATE mobile robot"

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

2.2.3 Linux Operating System: Linux [3] is an operating system identical to Windows and MacOS. In fact, one of the most popular open-source platforms on the planet, Android, is based on the Linux operating system. An operating system (OS) is software that manages all the hardware resources associated with desktop or laptop. The operating system manages communication between your software and your hardware. The software would not function without the operating system.

2.2.4 Ubuntu: Ubuntu [4] Desktop is a Linux distribution developed by Canonical. Linux distribution is an operating system developed from the Linux kernel (UNIX-like system created by Linus Torvalds in 1991). Linux distributions are free and open source. Various Linux distributions, such as Ubuntu or Linux Mint, are great alternatives to popular proprietary operating systems like Windows and macOS. Ubuntu is one of the most popular distributions due to simplicity. Ubuntu is one of the top choices among the users introduced to Linux. The Ubuntu server edition is commonly used for running on multiple internet servers.

2.2.5 Robot Operating System: ROS [5] (Robot Operating System) is an open-source software development framework for robotics applications. ROS offers a standard software platform to developers across industries helping in research and prototyping all the way through to deployment and production.

i. **ROS Melodic Morenia:** ROS Melodic Morenia [15] is the twelfth ROS distribution released on May 23rd, 2018. ROS Melodic Morenia was created to fully function on Ubuntu 18.04 LTS despite having support in other operating systems such as Mac OS X and Windows.

ii. **ROS Foxy Fitzroy:** ROS Foxy Fitzroy [16] is the first LTS (Long Term Support) of ROS2 targeting Ubuntu 20.04 (Focal Fossa). ROS Foxy introduces new features such as security enhancements via Data Distribution Service (DDS), new methods of communication with the lack of ROS Master, increased performance regardless of the network situation, multi-platform support (Linux, Windows, macOS).

2.2.6 Gazebo integration with ROS2: Gazebo is a stand-alone application which can be used independently of ROS or ROS 2. The integration of Gazebo with either ROS version is done through a set of packages called "gazebo_ros_pkgs". These packages provide a bridge between Gazebo's C++

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

API and transport system, and ROS 2 messages and services.The ROS 2 package gazebo_ros_pkgs is a metapackage which contains the following packages:

i. **“gazebo_dev”:** Provides a cmake configuration for the default version of Gazebo for the ROS distribution. So downstream packages can just depend on gazebo_dev instead of needing to find Gazebo by themselves.

ii. **“gazebo_msgs”:** Message and service data structures for interacting with Gazebo from ROS 2.

iii. **“gazebo_ros”:** Provides convenient C++ classes and functions which can be used by other plugins, such as **“gazebo_ros::Node”**, conversion, and testing utilities. It also provides some useful plugins.

iv. **“gazebo_plugins”:** A series of Gazebo plugins exposing sensors and other features to ROS 2. For example, **“gazebo_ros_camera”** publishes ROS 2 images, and **“gazebo_ros_diff_drive”** provides an interface for controlling and instrospecting differential drive robots through ROS 2.

2.2.7 3D Modelling: 3D models are an essential component of robot simulation as they represent the physical appearance and functionality of the robot. Creating accurate 3D models is critical for simulating the robot's movements, interactions with the environment, and testing various scenarios. The process of creating 3D models for robot simulation involves the clear understanding of the robot's physical appearance and functionality. This involves analyzing the robot's specifications, such as its dimensions, weight, and range of motion.

i. **Blender:** Blender [6] is a free open-source program provided by the Blender foundation. Blender is one of the fastest developing 3D toolkits currently available and one of the fastest growing online communities in the technology space. The primary use of Blender is to create three-dimensional objects and scenes using the 3D viewport and a suite of modeling tools that allow the user to create any object and any scenario that they want to create in 3D space. Basic objects and environments can be created using mesh objects constructed from three types of geometry. These are vertices, edges, and faces. By manipulating geometry, the user can control the form of any 3D object created. There are different

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

methods to create the objects in Blender and different toolsets to accommodate those methods.

ii. **Fusion360:** Fusion 360 [7] combines CAD and CAM applications into a single, well-integrated software. It includes all the tools needed to go from design to fabrication, without having to leave the platform. Fusion 360 has a complete selection of data translators for 50 different file types, which means it is easy to import files and get access to hundreds of posts from the Fusion 360 free post processor library to machine the product in Fusion 360's CAM environment.

2.2.8 3D Format for Gazebo: Gazebo can support various 3D file formats and there are many ways to import and visualize 3D models for Gazebo, enabling the creation of realistic virtual environments for robot simulations. By using these formats, the user can define the geometry, appearance, and physical properties of objects in the simulation, allowing for accurate and detailed simulations of robotic systems.

i. **.obj file:** OBJ files [8] are standard 3D formats that can be exported and imported by various 3D design programs. The OBJ file format was created in 1992 by Wavefront Technologies. It is a helpful file format as it is simple to work with and supports various software programs making it easy to share files between different applications. It enables users to represent complex or irregularly shaped objects by dividing their surface into small, triangular "tiles." This tessellation process makes it easier to manipulate and render the design since its can modify each tile separately from the rest. Another critical feature of OBJ files is their ability to specify the geometry of 3D objects and their surface properties, including texture mapping and shading. This versatility makes the OBJ file format robust for creating realistic renderings of complex three-dimensional scenes. OBJ files also contain information on free-form surface patches. These patches allow designers to create smooth surfaces free from distortions or seams, making them ideal for creating highly realistic textures such as skin or fabric. To create a model for Gazebo, the designers will need to convert the OBJ file into a format that is compatible with Gazebo, such as Collada (DAE) or STL.

ii. **.dae file:** DAE [9] file is a Digital Asset Exchange file format that is used for exchanging data between interactive 3D applications. This file

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content, and cite the document when use.

format is based on the COLLADA (Collaborative Design Activity) XML schema which is an open standard XML schema for the exchange of digital assets among graphics software applications. It has been adopted by ISO as a publicly available specification, ISO/pAS 17506. In the context of mobile robot simulation with Gazebo, DAE files are suitable for creating maps because of several reasons.

- DAE files are widely supported by many 3D modeling software applications, including Blender. Blender is a popular software application for creating 3D models, and it can export DAE files. This means that developers can create maps using Blender and export them as DAE files, which can then be imported into Gazebo for robot simulation.

- DAE files can contain not only 3D geometry but also additional information such as material properties, textures, and lighting. This means that maps created in Blender and exported as DAE files can include realistic textures and lighting, making the simulation more accurate and realistic.

- DAE files can be easily edited and modified. This means that if developers need to make changes to the map, they can do so using Blender and then export the map again as a DAE file. This allows for faster iteration and testing of different map configurations.

- DAE files are lightweight and can be easily loaded into Gazebo. This means that maps created using DAE files can be used in Gazebo without significant performance issues. Additionally, DAE files are compatible with Gazebo's terrain plugin, which allows developers to create more realistic maps with varied terrain.

iii. **.stl file:** STL (StereoLithography) are popular file formats used for 3D modeling and simulation. While they can be used for creating maps for robot simulation, they may not be as suitable as DAE files for a few reasons. STL files are typically used for creating 3D models for 3D printing. They contain only surface geometry information and do not include additional information such as material properties or textures. While STL files can be imported into Gazebo, they may not provide as

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

much detail or accuracy as DAE files, especially when it comes to lighting and textures.

2.2.9 Robot and Environment Modelling:

i. **URDF file**: URDF (Universal Robotic Description Format) is a file format used to describe the physical and kinematic properties of a robot in a standardized way. It is an XML file that contains information such as the links, joints, sensors, and visual representations of the robot. Procedures of describing a robot via URDF are as follows:

- Define the links that make up the robot. Each link has a name and can have a visual and collision representation. The visual representation is what the robot looks like, while the collision representation is what other objects see when colliding with the robot.

- define the joints that connect the links together. Joints describe how one link can move relative to another link, and they can be of various types such as revolute, prismatic, or fixed. Additionally, sensors and controllers can be added to the robot to control its movements and interactions with the environment.

URDF file format

```
<gazebo>
<plugin name="differential_drive_controller"
filename="libdiffdrive_plugin.so">
... plugin parameters ...
</plugin>
</gazebo>
```

ii. **SDF file**: SDF (Simulation Description Format) file format is an XML-based file format used to describe models in Gazebo, the open-source robot simulation software. SDF files define the structure, properties, and behavior of entities within the simulation, including robots, objects, sensors, and environments. The process of conversion from URDF to SDF can be easily done by adding the plugins called **"gazebo_plugins"** into URDF file. The gazebo plugins can attach to ROS messages and

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

service calls the sensor outputs and driving motor inputs, i.e., the gazebo plugins create a complete interface (Topic) between ROS and Gazebo.

SDF file from **"gazebo_plugins"**

```
<model name="robot_model">
<plugin name="differential_drive_controller"
filename="libdiffdrive_plugin.so">
... plugin parameters ...
</plugin>
</model>
```

2.2.10 Mobile Robot Control: Mobile robot control involves the techniques and algorithms used to govern the movement and behavior of a mobile robot. It encompasses various aspects such as path planning, motion control, obstacle avoidance, and localization. The goal is to enable the robot to navigate its environment effectively, perform tasks, and achieve desired objectives.

i. **Simulation**: Mobile robot simulation refers to the process of simulating the behavior, movement, and interaction of a mobile robot in a virtual environment. It allows researchers, engineers, and developers to test and evaluate robot algorithms, control strategies, perception systems, and navigation techniques in a safe and controlled manner before deploying them on real robots.

- **Robot Description Format**: Gazebo supports popular robot description formats such as URDF (Unified Robot Description Format) and SDF (Simulation Description Format). These formats allow users to define the robot's physical structure, including its links, joints, sensors, and visual properties. This information is then used by Gazebo for accurate simulation and visualization of the robot.

ii. **Locomotion and sensing**

- **Gazebo ROS sensors plugin**: Gazebo ROS provides a range of sensors plugins that can be used for locomotion and sensing in mobile robot simulations. These plugins integrate the capabilities of Gazebo's physics simulation and ROS's sensor interfaces, allowing users to simulate and interact with various sensors

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

commonly used in mobile robotics. Here are some commonly used Gazebo ROS sensors plugins.

iii. **Basic Operation**

- **Topic Monitoring**: Topic monitoring in the context of mobile robot simulation refers to the process of observing and analyzing the data published on specific ROS topics related to the robot's state, sensors, or other relevant information. It involves subscribing to specific topics and extracting relevant data for monitoring and analysis purposes.

  - **Topic Selection:** Identify the ROS topics that contain the information you want to monitor. These topics can include sensor data, robot state information, diagnostic messages, or any other data relevant to the application.

  - **Subscription:** Create one ROS node or script that subscribes to the selected topics. This involves specifying the topic names, message types, and defining callback functions to process the received data.

  - **Data Analysis and Visualization:** Process the received data in the callback functions to extract relevant information. Perform any required analysis or computations on the data to derive insights or make decisions. You can also visualize the data using various ROS visualization tools, such as RViz or custom visualization nodes.

  - **Logging and Recording:** Optionally, log the monitored data to a file for further analysis or documentation. ROS provides logging capabilities that allow you to record data from specific topics to a file for offline analysis or replay.

  - **Real-Time Monitoring:** In some cases, real-time monitoring of topics is crucial, especially in applications that require immediate feedback or control. Ensure that the topic monitoring system has minimal latency and can

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

handle the data processing and visualization within the desired time constraints.

- **Teleoperation**: Teleoperation is a method of controlling a mobile robot remotely using human input. In the context of ROS (Robot Operating System), teleoperation is often achieved by sending velocity commands to the robot's base using the **cmd_vel** topic.

- **SLAM (mapping)**: Mapping using Cartographer is a technique employed in mobile robot simulation and navigation to create a map of the robot's environment. Cartographer is a powerful open-source library developed by Google that utilizes simultaneous localization and mapping (SLAM) algorithms to generate accurate and detailed maps.

  - **Sensor Data Acquisition:** Cartographer relies on sensor data, particularly from 2D lidar sensors, to perceive the surrounding environment. The robot's sensors collect range measurements and pose information as it moves through the environment.

  - **Data Preprocessing**: Before feeding the sensor data to Cartographer, it is essential to preprocess it. This step involves removing noise, filtering outliers, and synchronizing the data from multiple sensors if present (e.g., lidar and IMU data fusion).

  - **Configuration Setup:** Configure Cartographer by specifying the parameters and settings according to your robot's sensors, dimensions, and desired map output. This includes defining sensor extrinsics (e.g., sensor-to-robot transformation) and tuning various algorithmic parameters.

  - **Online SLAM:** Cartographer performs online SLAM to estimate the robot's pose (localization) and simultaneously

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

build a map of the environment. It employs advanced algorithms such as scan matching, loop closure detection, and pose graph optimization to ensure accurate mapping even in challenging scenarios.

- **Map Generation:** As the robot explores the environment, Cartographer incrementally builds a map representation. The map can be a 2D occupancy grid, a 3D point cloud, or a combination of both, depending on the sensor data and configuration.

- **Map Post-Processing**: Once the mapping process is complete, post-processing steps may be applied to refine the map further. This can include map filtering, map compression, or merging multiple maps acquired from different sessions.
- **Map Visualization**: The generated map can be visualized using tools such as RViz, Gazebo, or custom visualization software. This allows users to inspect the map, assess its quality, and make any necessary adjustments to improve the mapping results.

▪ **Navigation**: Navigation using nav2 (Navigation2) is a powerful framework for autonomous robot navigation in ROS2 (Robot Operating System 2). It provides a comprehensive set of capabilities and tools for path planning, obstacle avoidance, and motion control, allowing mobile robots to navigate safely and efficiently in complex environments.

- **Map Representation**: Navigation begins with a map of the environment, typically created using mapping techniques like SLAM. The map can be in the form of a 2D occupancy grid or a 3D point cloud, depending on the sensor data and requirements.

- **Localization**: Before navigation can take place, the robot needs to estimate its own position within the map. Localization algorithms, such as Monte Carlo Localization (MCL) or AMCL (Adaptive Monte Carlo Localization),

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

are used to determine the robot's pose (position and orientation) based on sensor measurements and the map.

- **Path Planning**: Nav2 provides advanced path planning algorithms to generate collision-free paths for the robot. Common algorithms include Dijkstra's algorithm, A* search, or the DWA (Dynamic Window Approach) algorithm. The planner considers the robot's current pose, the map, and any specified goals or constraints to compute an optimal or feasible path.

- **Obstacle Avoidance**: During navigation, the robot continuously monitors its surroundings for obstacles using sensors such as lidar or depth cameras. Nav2 employs obstacle avoidance techniques to dynamically adjust the robot's path or velocity to avoid collisions with detected obstacles. This can include reactive approaches, such as local obstacle avoidance, as well as global replanning if necessary.

- **Motion Control**: Once a path is planned and obstacles are avoided, the robot needs to execute the desired motion. Nav2 interfaces with the robot's low-level control system, such as ROS controllers or hardware interfaces, to send velocity commands and control the robot's actuators, such as wheels or motors. The control system ensures the robot follows the planned path accurately.

- **ROS2 Navigation Stack**: Nav2 is built upon the ROS2 Navigation Stack, which provides a modular and configurable framework for navigation. The stack consists of various components, including the map server, localization, path planner, and controller, which work together to achieve autonomous navigation.

- **Configuration and Customization**: Nav2 allows extensive configuration and customization to adapt to different robot platforms and application scenarios. Parameters can be tuned to adjust navigation behavior,

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

such as the planner's parameters or sensor settings. Additionally, custom plugins and behaviors can be added to enhance navigation capabilities.

2.2.11 ROS package migration: The disadvantage of open-source software is the life cycle. The current ROS1 version of the Seagate Mk. II is ROS Melodic Morenia. ROS Melodic would reach EOL in April 2023 along with Ubuntu 18.04 Bionic Beaver. The migration is the adjustment of the old packages to be compatible with the newer version of ROS2 while maintaining the same functionality. Migration also ensures the further development of the robot utilizing the new features of ROS2.

2.2.12 ROS1 VS ROS2:

i. **Distribution:** ROS Noetic (release date: 2020) is the last ROS1 version. The final ROS1 version's main goal is to provide Python3 support for developers/organizations who need to continue working with ROS1 for a while. ROS Noetic's EOL (End of Life) is scheduled for 2025. After that, ROS1 will not have further update. For ROS2, from the LTS (Long Term Support) version Foxy Fitzroy (release date: 2020), a new ROS2 version is released every year. Developers can work with the stable ROS2 distros utilizing the core capabilities while some third-party plugins to update.

ii. **Node Writing**

- **ROS API – rclcpp, rclpy:** ROS1 commonly uses separate libraries called "roscpp" for C++ and "rospy" for Python for writing nodes. These libraries were developed independently and from scratch, which means different APIs. Some features were only available in one library and not in the other. ROS2 takes a different approach by introducing more layers. ROS2 has a base library called "rcl," written in C, which contains all the core features of ROS2. However, developers don't directly use the "rcl" library in their programs. Instead, they use client libraries built on top of "rcl," such as "rclcpp" for C++ and "rclpy" for Python. New functionality only needs to be implemented within the "rcl" library, and the client libraries on top of it provide the necessary bindings. For developers, this means:

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

- The API between "rclcpp" and "rclpy" is more similar compared to the API between "roscpp" and "rospy" in ROS1.

- It becomes easier to create and use client libraries in other programming languages, like "rclnodejs" or "rcljava." Developers don't need to start from scratch; they only need to create a C binding with "rcl." This ensures that all clients in different languages have a similar API.

- With the separation between the core features in "rcl" and the client libraries, when a new core feature is released, it becomes available in different languages sooner. Developers don't have to wait as long to use new features in their preferred programming language.

- **Python and Cpp versions:** In ROS1, there have been notable developments regarding the supported versions of Python and C++. Python 2, which was widely used in the past, is no longer supported. However, to make the transition smoother, Python 2 is still supported for Ubuntu 18 and ROS1 Melodic until their End-of-Life (EOL) in 2023. The industry trend now favors Python 3 as the recommended version. In terms of C++, ROS1 primarily used C++ 98, but later versions allowed the use of C++ 11/14 if it didn't cause conflicts with other dependencies. In ROS2, the newer version of the Robot Operating System, there have been advancements in the supported versions of Python and C++. Python 2 is no longer supported, aligning with the industry shift towards Python 3. ROS2 supports Python 3 across all versions. Regarding C++, ROS2 supports C++ 11 and 14 by default, with plans to include support for C++ 17 in the future. This transition to newer C++ versions brings useful functionalities, improving development efficiency, speed, and safety. These developments contribute to making C++ more accessible and popular as a programming language.

- **OOP (Object-Oriented Programming)**: there are distinct approaches between ROS1 and ROS2. In ROS1, no specific

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

structure dictates how node functionalities should be implemented. Developers have the freedom to add callback functions anywhere in their program or employ **object-oriented programming (OOP)** if desired. Consequently, each implementation can be unique. In contrast, ROS2 introduces a convention for node writing. Developers are required to create a class that inherits from the Node object, such as "**rclcpp::Node**" in C++ or **"rclpy.node.Node**" in Python. The class serves as a container for all ROS2 functionalities associated with the node. In ROS1, a node is closely tied to an executable. To enable writing multiple nodes within a single executable and facilitate intra-process communication, ROS1 introduced Nodelets. However, in ROS2, this concept has been integrated directly into the core framework and is now referred to as "components." Components function as slightly modified node classes, still leveraging the principles of OOP. Multiple nodes can be managed from the same executable using components, which can be started from a launch file, terminal, or as an executable. Furthermore, intra-process communication can be activated to eliminate communication overhead in ROS2.

- **Lifecycled nodes**: ROS2 introduces lifecycled nodes, which offer many states: unconfigured, inactive, active, and finalized. These states prove to be useful when a setup phase is required before executing a node's core functionalities. When starting a lifecycled node, it begins in an unconfigured state. Through the ROS interface (typically ROS2 services), state transitions can be requested, triggering specific function callbacks within the node. Lifecycled nodes facilitate a clear separation of these steps: allocating memory for essential objects, initializing sensor communication, and executing the reading loop for data publication. This feature enables a more structured and more controlled approach to node initialization, leading to more modularity and organization while enhancing the reliability and functionality of ROS2-based systems.

- **Launch files**: Launch files allow the user to start all nodes from one file. It can start a standard node, a component, a lifecycled node, it can also add arguments, parameters, and many other options. In ROS1, launch files were typically written in XML.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

However, in ROS2, Python is used for writing launch files. ROS2 provides an API that enables node launching, configuration file retrieval, parameter addition, and more. This Python-based approach led to more enhanced customization capabilities for launch files compared to XML.

iii. **Communication**:

- **No ROS Master**: ROS1 always starts a ROS master before running a node. The ROS1 master will function as a DNS server for the nodes to communicate with each other. In ROS2, ROS master is no longer used as ROS2 is a decentralized system. Each node can discover other nodes. A node can be launched without connecting to the master. The feature allows developers to create a fully distributed system. Each node is independent and not tight to the global master. When creating a multi-machine ROS2 application, there is no need to define one machine as the "master". Each machine will be independent and able to start on its own, connect and disconnect with each other, with less setup than in ROS1. This also means that ROS2 has no single point of failure (master), this increases the fault tolerance of the system.

- **Parameters**: In ROS1, parameters are handled by the parameter server, which is itself handled by the ROS master. In ROS2, no more ROS master means no more (global) parameter server. The concept of parameters has been completely changed. There is no global parameter anymore. Each parameter is specific to a node. A node declares and manages its own parameters, and those parameters are destroyed when the node is killed. This architecture is like having individual parameter servers for each node. When starting a node in ROS2, several ROS2 services are created, enabling interaction with the node's parameters from the terminal or other nodes. Additionally, modifying a node's parameters after creation is made simple through the utilization of a parameter callback.

- **Service**: In ROS1, services are synchronous. When the service client sends a request to the server, it is stuck until the server responds (or fails). In ROS2, services are asynchronous. When

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

the service gets called, a callback function can be added which will be triggered when the server responds. In the meantime, the main thread is not stuck. ROS2 can also use services synchronously.

- **Action**: In ROS1, actions were added later as an extension to solve limitations in synchronous services. They were built on top of ROS1 topics. However, in ROS2, actions are now integrated into the core framework. ROS2 actions still use topics for feedback and goal status, but they also employ (asynchronous) services for goal setting, cancellation, and result requests. Additionally, ROS2 introduces a dedicated command-line tool for sending action goals directly from the terminal, similar to service requests.

- **Message, Service, and Action definition**: The creation of definitions for messages, services, and actions are similar in ROS1 and ROS2. These definitions are placed in designated folders: msg/ for messages, srv/ for services, and action/ for actions. In ROS2, there is a slight difference once the definitions are compiled a namespace is added to the paths.:
    - Message: msg/…
    - Services: srv/…
    - Actions: action/…

  For example, in the ROS2 package named my_robot_msgs, and inside of the package there is a message named Temperature, plus a service named ActivateButton. In the node's code the import statement will be:

    - /my_robot_msgs/msg/Temperature.
    - /my_robot_msgs/srv/ActivateButton.

  Compare to ROS1

    - /my_robot_msgs/Temperature.
    - /my_robot_msgs/ActivateButton.

  This reduces confusion and makes the separation clearer between all 3 types of communication.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

iv. **Packages, workspace, and environment:**

- **Building a node**: The build system in ROS1 is catkin. "**catkin_make**" or "**catkin build**" is used to build and install your packages. In ROS2, Catkin is replaced with Ament. Ament is the new building system, and the colcon command line tool is used instead. To compile, the users use the command "**colcon build**" in the ROS2 workspace.

- **Command Line Tool**: Most of the command line tools are like ROS1 and ROS2. The name of the tools, and some options are different, but otherwise there is no significant difference when you use them. For example,

  - To list all topics
    - ROS1: "rostopic list"
    - ROS2: ros2 topic list.
  - To list all services
    - ROS1: "rosservice"
    - ROS2: "ros2 service"
  - To run the source code
    - ROS1: "rosrun"
    - ROS2: "ros2 run"

  The ROS2 version of the command line tools is changed from starting with "ros" to starting with "ros2" and the rest of the command line remains the same.

- **C++ and Python packages**: With ROS1, the package did not need to be specified as Python or C++ package. ROS2 makes the difference between a C++ and a Python package. When creating the package from the command line, the package build type is specified as follows:
  - **ament_cmake**
  - **ament_python**.

  Depending on the argument, the package architecture is not always the same. For C++ package, it is like ROS1. The **CMakeLists.txt** still exists. The instructions are changed to

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content and cite the document when use.

use **ament_cmake** instead of **catkin**. For Python package, **setup.py** and **setup.cfg** are added. The **setup.py** replaces the **CMakeLists.txt**. Python scripts are executable. But in case ROS2 command line tools or a launch file is preferred, package installation via "**colcon build**" is required.

2.2.13 ROS OS Support: ROS1's main target is Ubuntu. ROS2 installation can be done on Ubuntu, MacOS, and Windows 10 due to the multiple OS support. The multiple OS support makes ROS2 more accessible and more embeddable in many applications. For example, one project could have a mobile robot with Raspberry Pi and Ubuntu, and another computer using Windows for a 3D simulation tool and a driver node for a camera scanning the scene.

2.2.14 ROS Package Components: A package is an organizational unit for a ROS2 code. If developers want to install the code or share it with others, then the code needs to be organized in a package. Packages enable any ROS2 code to have easier installation.

i. **CMakeLists.txt**: The file CMakeLists.txt is the input to the colcon build system for building software packages. Any colcon-compliant package contains one or more CMakeLists.txt file that describes how to build the code and where to install it.

ii. **package.xml**: The package manifest is an XML file called package.xml that must be included with any colcon-compliant package's root folder. Package.xml defines the properties of a package such as the package name, version numbers, authors, maintainers, and dependencies on other catkin packages.

iii. **Source code**: The main code no matter the format (.cpp/ .py) is required to be included in the "src" directory. It is the source code for the package.

iv. **Message/Service**: Message and Service files (.msg/.srv) is called an Interface in ROS2.

v. **Launch file**: ROS 2 Launch files allow the developers to start up and configure several executables containing ROS 2 nodes simultaneously. In ROS2, the launch file can be written in .xml, .py, or. yaml format.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

2.3 Related Research

- **Comprehensive Simulation of Quadrotor UAVs Using ROS and Gazebo:** Johannes Meyer, Alexander Sendobry, Stefan Kohlbrecher, Uwe Klingauf & Oskar von Stryk

- **A Survey on Mobile Robot Simulation Techniques by M. Alaghband** et al. (2018). This survey provides a comprehensive overview of mobile robot simulation techniques, including kinematic and dynamic modeling, sensor modeling, and control system modeling.

- **Simulation of Mobile Robots with ROS: From Gazebo to Real World by S. Koenig et al. (2018).** This paper describes a simulation framework for mobile robots using the Robot Operating System (ROS) and Gazebo simulator. The authors demonstrate the effectiveness of their approach with experiments on a real robot.

King Mongkut's Institute of Technology Ladkrabang

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

CHAPTER 3
METHODOLOGY

3.1 Introduction

This research aimed to investigate the impact that simulations have on the real-project deployment, and about the method of ROS package migration. The following will provide a detailed description about the simulation and migration method.

3.2 Simulation

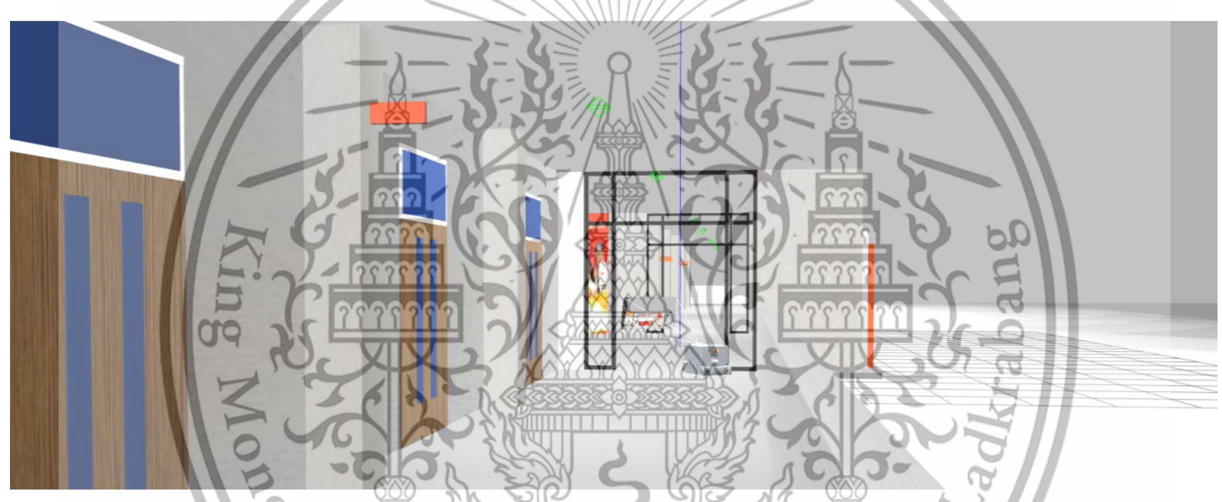


Figure 3. 1 Seagate Robot MK II Simulation

In the realm of mobile robot development, simulation plays a crucial role in testing algorithms and validating the performance of various functionalities before real-world deployment. This research focuses on implementing an experiment using simulation to assess the capabilities of the Seagate mobile robot. The methodology involves creating highly realistic 3D maps that closely resemble real-world environments. Through this simulation-based approach, by aiming to thoroughly test the sensor outputs of the Seagate robot, including the motor drive, lidar-based mapping, and navigation using the ROS Navigation Stack (nav2). Additionally, we will utilize 2D and 3D cameras for object detection and recognition.

The creation of accurate and representative 3D maps is pivotal in ensuring a realistic simulation environment. By closely mimicking the real world, we can

King Mongkut's Institute of Technology Ladkrabang
Forbidden to modify the content, and cite the document when use.

effectively evaluate the performance of the Seagate robot's sensors and algorithms. The maps will incorporate details such as obstacles, terrain variations, and structural elements found in typical operational scenarios. This comprehensive simulation environment allows us to validate the Seagate robot's response and behavior under various conditions, ensuring robustness and reliability.

The sensor outputs of the Seagate robot play a crucial role in its perception and decision-making processes. The motor drive system is responsible for accurately controlling the robot's movements and velocity. Through simulation, we can thoroughly assess the motor drive system's performance and evaluate its ability to navigate through different environments while maintaining desired motion characteristics.

In summary, the experiment implementation and methodology leverage simulation to create 3D maps that closely resemble real-world environments. We aim to thoroughly test the Seagate mobile robot's sensor outputs, including the motor drive system, lidar mapping, and object detection using 2D and 3D cameras. By utilizing this simulation-based approach, we can ensure the reliability and robustness of the Seagate robot's performance, facilitating its successful real-world deployment.

3.2.1 **Specify the Simulation's environment:**

For the first part of simulation start with the step of deciding to choose an environment for creating the 3d map for the Seagate mobile robot simulation. We decided to choose three real-world-places that take place in KMITL.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

i. **HM Robotics Laboratory:**

First is HM Robotics Laboratory. The HM Robotics Laboratory is the main Laboratory for all Robotics and AI students to learn, experiment and work together located at HM Building 1st floor.

The reason for choosing HM Robotics Laboratory to be the simulation map is because the students in the major of Robotics and AI are familiar with this place. The lab also represents the image of Robotics and AI.

Figure 3. 2 HM Robotics Laboratory

King Mongkut's Institute of Technology Ladkrabang

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content, and cite the document when use.

ii. **HM Building 6th floor:**

The second place is HM Building 6th floor. The HM Building 6th floor is another landmark of the HM Building. There are included 3 lecture rooms, 1 School of International & Interdisciplinary Engineering Programs (SIIE) office and 1 co-working space.

Figure 3. 3 HM Building 6th floor

Mongkut's Institute of Technology Ladkrabang

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

iii. E12 Building 12th Floor:

E12 Building is the main lecture, meeting, and event for KMITL Engineering students. In the few years, The E12 building will be renovate and Robotics and AI majors are planning to build the new robotics laboratory on the 12th floor. So, it is necessary to create a simulation map for Robotics and AI students in the future.

Figure 3. 4 E12 Building 12th floor

King Mongkut's Institute of Technology Ladkrabang

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content and cite the document when use.

3.2.2 Blueprint and Measurement for 3D creations:

After deciding which places to create a 3D map, the next step is to request for the building & laboratory blueprint. The blueprint is use for getting the exact scale of the big part of the model such as rooms and floors.

Figure 3. 5 Laboratory & Building floor blueprints

King Mongkut's Institute of Technology Ladkrabang

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

For the small part such as banners, laboratory doors, etc. Man-measurement is another tool using for the same purpose.

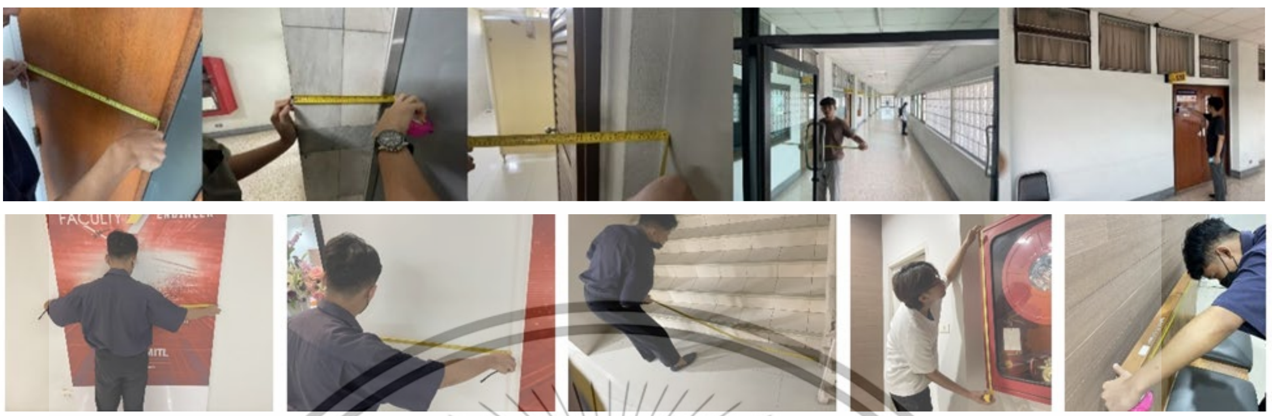


Figure 3. 6 Measurement

3.2.3 Create 3D map: The 3D map of all 3 places mentioned earlier is generated via using Blender and Fusion360 program by the steps:

1. **Design the Map Layout**: Determine the layout and structure of the map. In this case we based on the blueprint and measurement of the real environment. Consider the environment, obstacles, and any specific features such as robot arms, television, and banners.

2. **Create the Terrain**: Start by creating the ground or terrain of the map. every Blendr project starts with creating a square mesh and modifying it to be a ground.

3. **Add Objects and Structures**: Add objects or any other elements into the map.

4. **Define Collision Boundaries**: Set up collision boundaries for objects that should interact with the robot. This will allow the robot to detect and avoid obstacles. For those 3 maps we decided that the collision boundaries are mainly walls and windows.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

5. **Assign Materials and Textures**: Apply materials and textures to the objects to give them realistic appearances. The material properties can be configured by Blender's material editor to create and assign materials, and UV unwrap the objects to apply textures.

6. **Set up Lighting**: Add light sources to the map to illuminate the scene. Experiment with several types of lighting, such as sunlight or artificial lights, to achieve the desired atmosphere.

7. **Fine-tune the Environment**: Adjust the overall look and feel of the map by fine-tuning the materials, lighting, and other visual elements. You can also add details like props, decals, or particle effects to enhance realism.

8. **Export the Map**: Once satisfied with the map, the last step is to export it in a format suitable for the robot simulation. Blender supports various file formats, such as OBJ, STL, or COLLADA, which can be imported into simulation software (Gazebo).

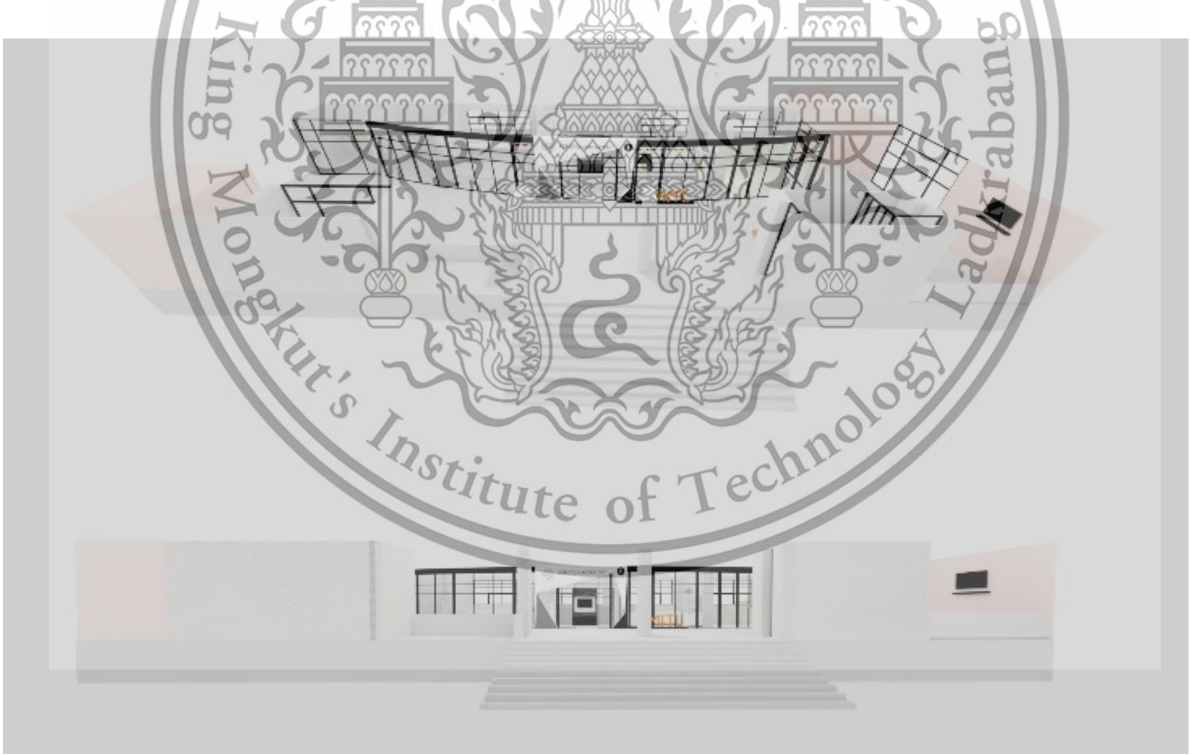


Figure 3. 7 HM Robotics Laboratory 3D map

King Mongkut's Institute of Technology Ladkrabang
This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

3.2.4 Seagate robot for simulation:

Creating a simulation environment for the Seagate mobile robot in ROS2 Gazebo involves several essential steps. The goal is to replicate the robot's behavior and sensor outputs in a simulated setting that closely resembles the real world. Simulation enables testing and validation of the robot's algorithms, control systems, and perception capabilities before deploying in actual environments. The general process of creating a Seagate mobile robot simulation in ROS2 Gazebo is as follows:

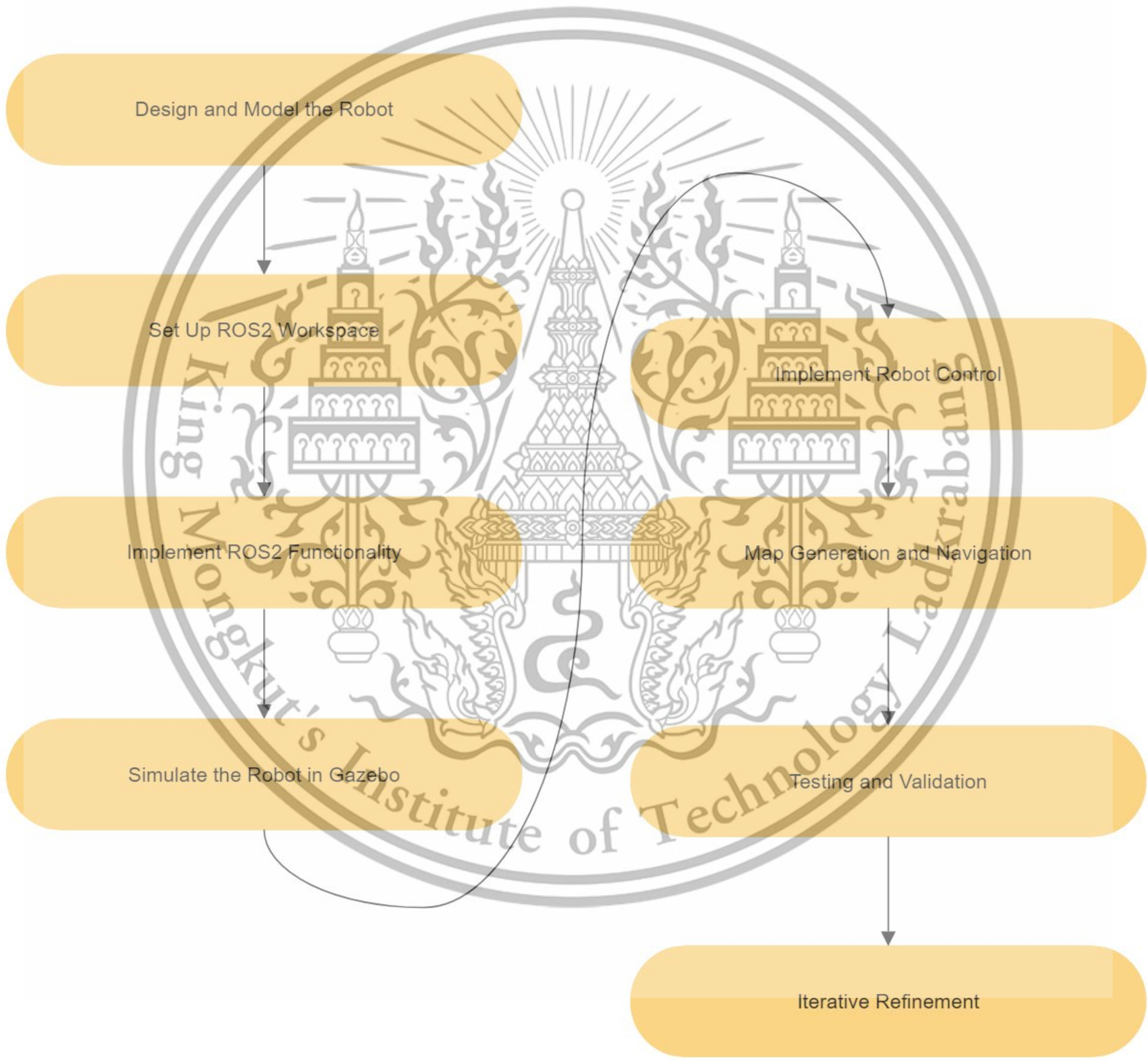


Figure 3. 8 3D robot simulation working flow

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Creating a Seagate Mobile Robot Simulation in ROS2 Gazebo involves several steps as follows:

1. **Design and Model the Seagate Mobile Robot**:

   - Begin by creating a detailed 3D model of the Seagate mobile robot using CAD software or ROS-compatible modeling tools like URDF (Unified Robot Description Format) to design and model the Seagate Mobile Robot in a simulation environment, utilizing the 3D models in file formats such as "**.dae**" (Collada) or "**.stl**" (stereolithography). These 3D models are used to define both the visual representation and collision properties of the robot within the simulation.

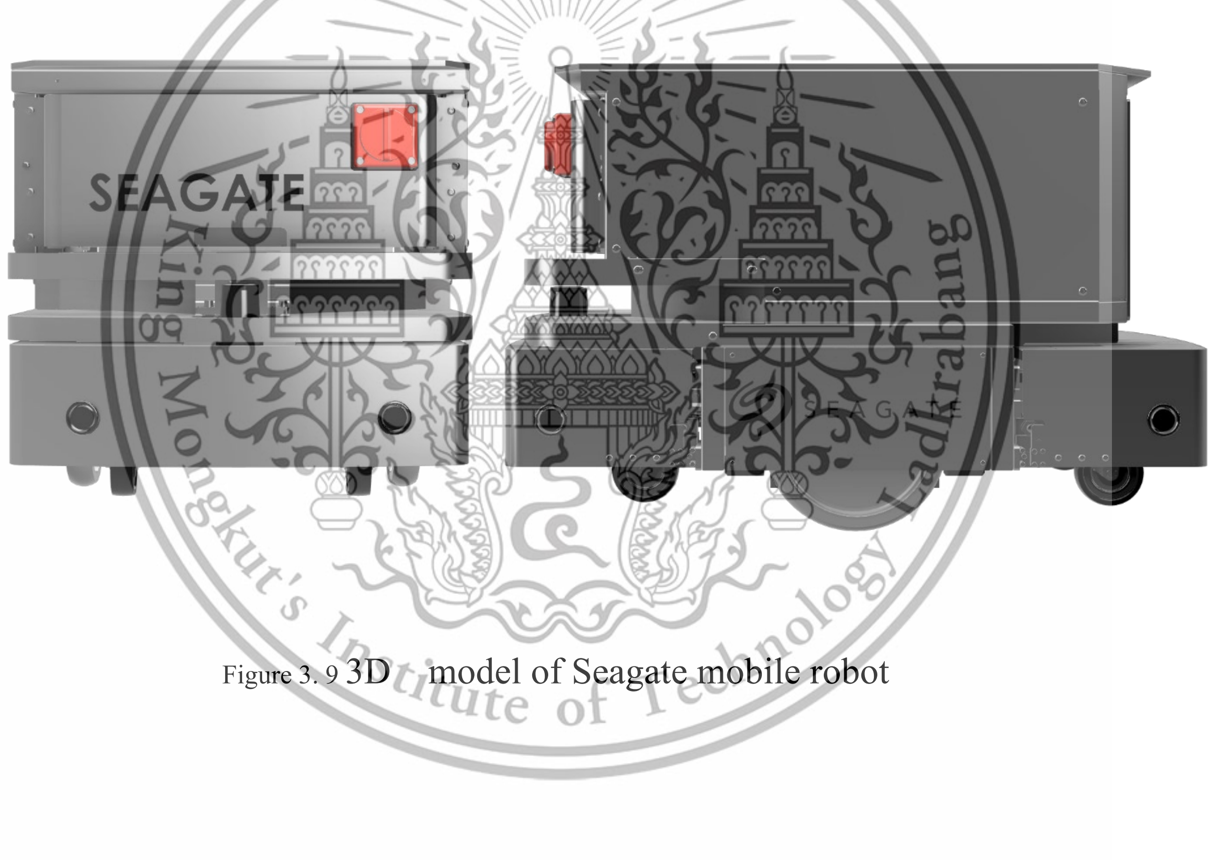


Figure 3. 9 3D model of Seagate mobile robot

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

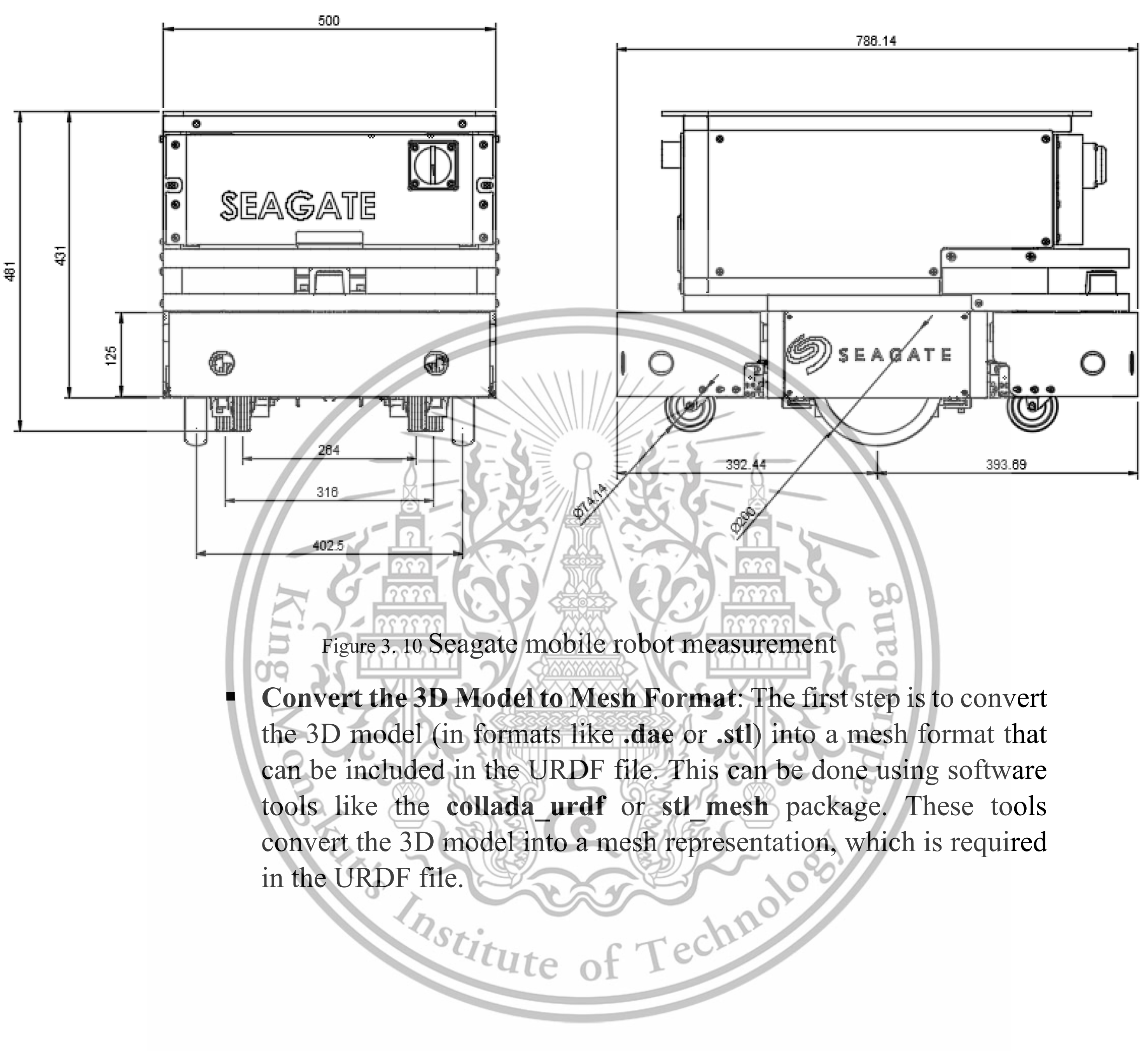


Figure 3. 10 Seagate mobile robot measurement

- **Convert the 3D Model to Mesh Format**: The first step is to convert the 3D model (in formats like **.dae** or **.stl**) into a mesh format that can be included in the URDF file. This can be done using software tools like the **collada_urdf** or **stl_mesh** package. These tools convert the 3D model into a mesh representation, which is required in the URDF file.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

```xml
    <collision name="base_collision">
      <pose>0.023314 0 0.279242 0 0 0</pose>
      <geometry>
        <mesh>
          <uri>model://turtlebot3_seagate/meshes/seagate_nocolor.stl</uri>
          <scale>1 1 1</scale>
        </mesh>
      </geometry>
    </collision>

    <visual name="base_visual">
      <pose>0 0.002 0 0 0 0</pose>
      <geometry>
        <mesh>
          <uri>model://turtlebot3_seagate/meshes/seagate.dae</uri>
          <scale>1 1 1</scale>
        </mesh>
      </geometry>
    </visual>
  </link>
```

Figure 3. 11 Seagate model with visual and collision representation

King Mongkut's Institute of Technology Ladkrabang

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

- **Define the URDF Structure**: Open a text editor and create a new URDF file. Begin by defining the overall structure of the robot in the n URDF file including the robot's name, base link, and any other joint or link connections.

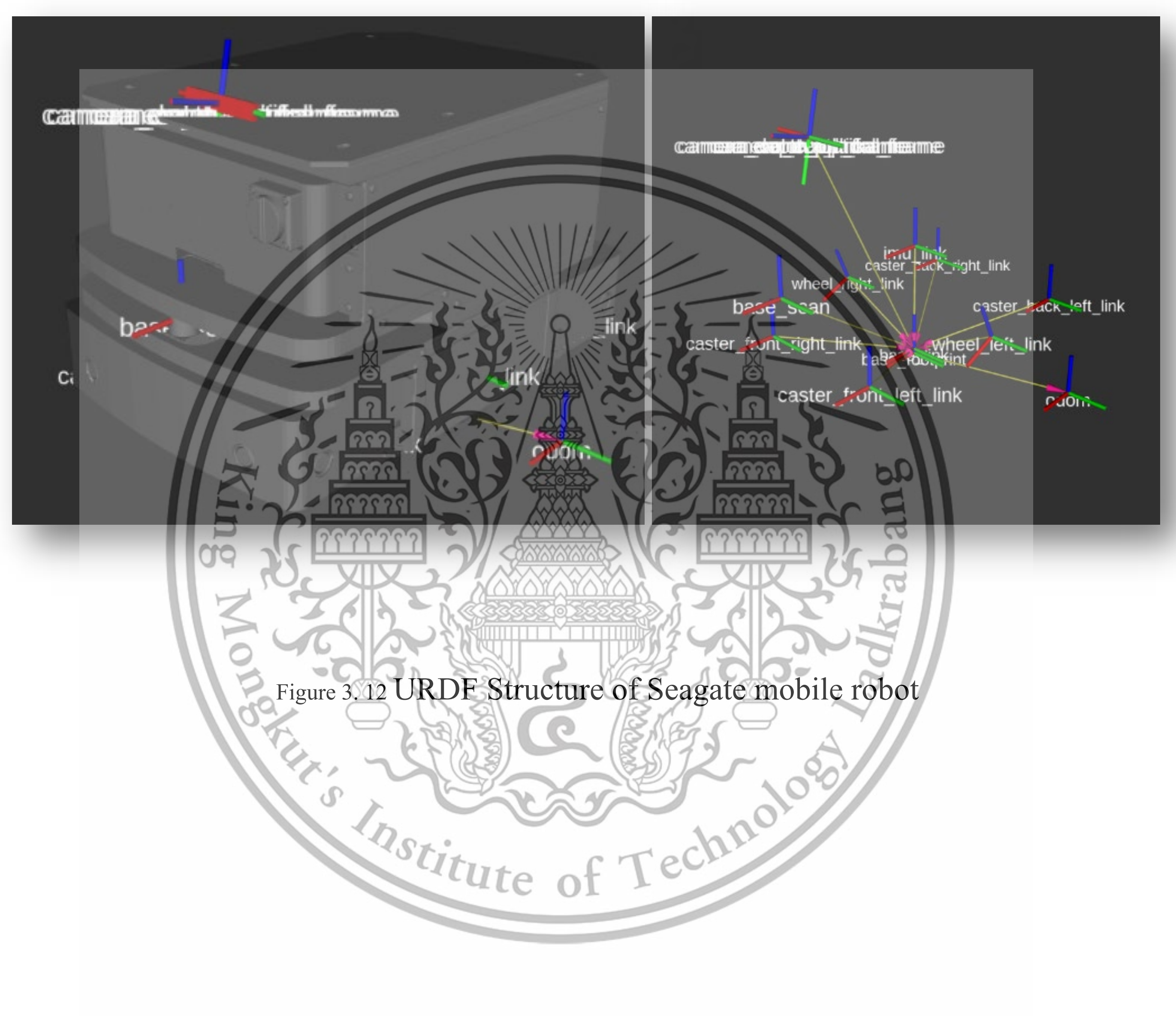


Figure 3. 12 URDF Structure of Seagate mobile robot

King Mongkut's Institute of Technology Ladkrabang

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

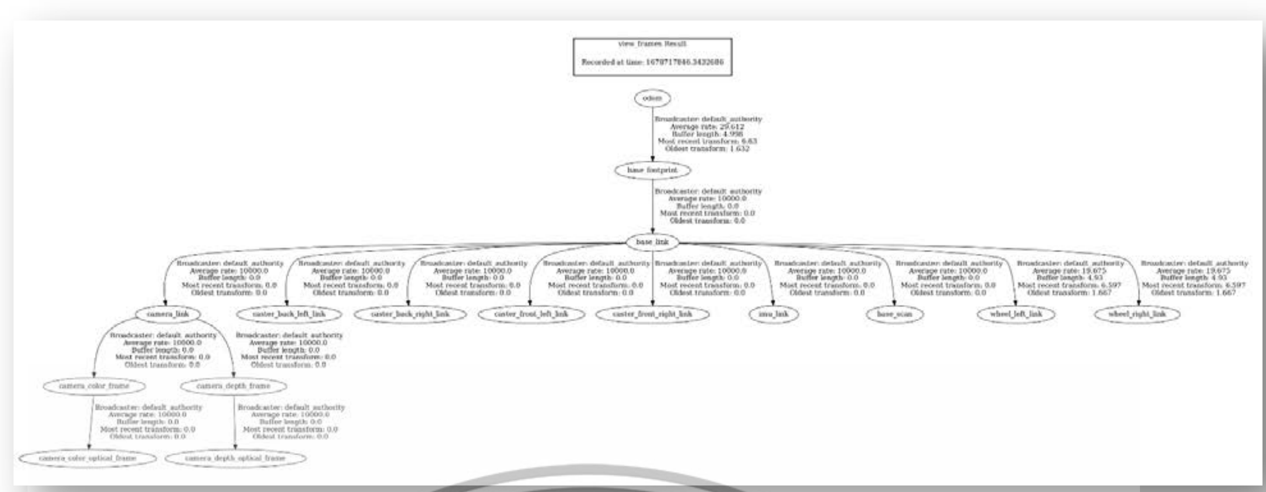


Figure 3. 13 Seagate mobile robot's TF2 frame

- **Physical Properties Definition**: Define the robot's physical properties, such as dimensions, mass, and inertial properties, in the URDF file.

  - **Visual Representation**: The visual representation of the robot is important for visualizing its appearance in the simulation. We can use 3D modeling software, such as Blender or SolidWorks, to create or modify a 3D model of the Seagate robot. This model should accurately represent the robot's physical structure, including its shape, size, and any distinctive features.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

```xml
    <visual name="wheel_left_visual">
      <pose>0 0.20 0.101902 0 0 0</pose>
      <geometry>
        <mesh>
          <uri>model://turtlebot3_seagate/meshes/wheel.dae</uri>
          <scale>1 1 1</scale>
        </mesh>
      </geometry>
    </visual>
  </link>
```

Figure 3. 14 Visual representation of wheel_left

- **Collision Properties**: In addition to the visual representation, defining the collision properties of the robot is crucial for ensuring accurate interaction with the environment and other objects in the simulation. To achieve this, we need to create a separate collision model that approximates the robot's physical boundaries.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

```
<link name="wheel_left_link">
  <collision name="wheel_right_collision">
    <pose>0 -0.20 0.101902 1.5707 0 0</pose>
    <geometry>
      <cylinder>
        <radius>0.1</radius>
        <length>0.04</length>
      </cylinder>
    </geometry>
    </surface>
  </collision>
```

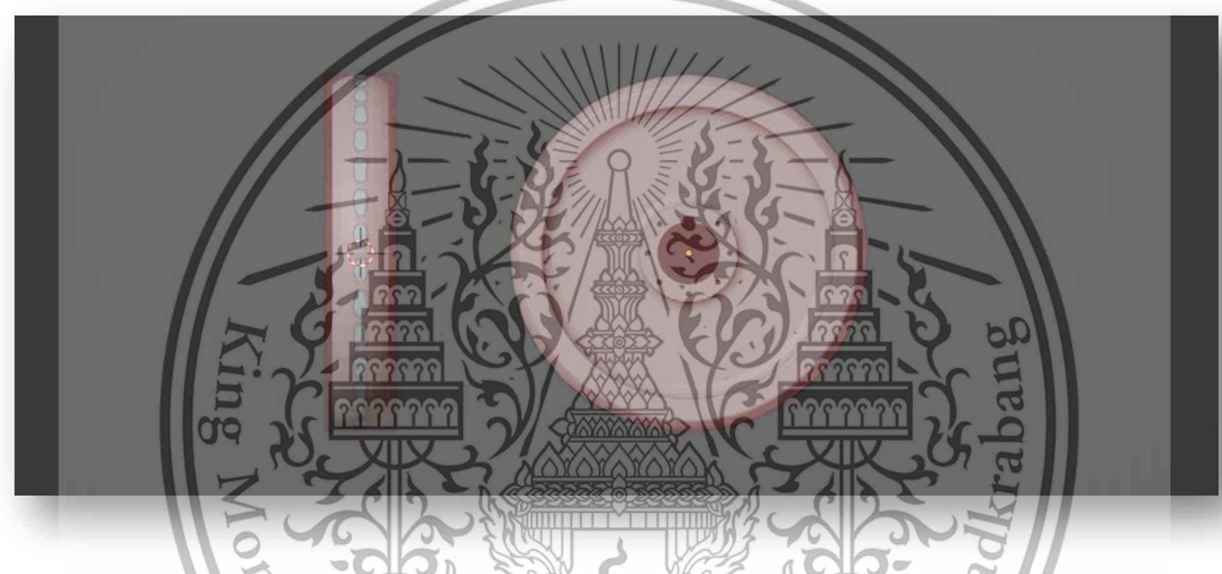


Figure 3. 15 collision representation of wheel_left

In some cases, the visual model can also be used as the collision model if the visual model accurately represents the robot's collision boundaries. However, collision models are commonly simplified to speed up simulations and improve computational efficiency. Simplified collision models are typically represented by a collection of basic geometric shapes like boxes, cylinders, or spheres that closely envelop the robot's physical structure. i.e

Collision wheel model using collision as cylinder

```
  <collision name="wheel_right_collision">
      <cylinder>
        <radius>0.1</radius>
        <length>0.04</length>
      </cylinder>
```

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

- **Joint Properties Definition**: For each joint in the robot's structure, define its properties within the URDF file including the specified the joint type (e.g., revolute, prismatic), the joint's origin and axis, and any joint limits or dynamics.

```
<joint name="wheel_right_joint" type="revolute">
  <parent>base_link</parent>
  <child>wheel_right_link</child>
  <pose>0 -0.20 0.101902 0 0 0</pose>
  <axis>
    <xyz>0 1 0</xyz>
  </axis>
</joint>
```

- **Incorporating the 3D Model in URDF**:In the URDF file, reference the visual and collision elements previously defined for each link. Associate the visual and collision elements with their respective links and joints by using the **<visual>** and **<collision>** tags within the **<link>** and **<joint>** tags including the necessary components, such as wheels, sensors (lidar, cameras), and any other specific hardware present on the real robot.

- **Include Additional Robot Parameters**: Depending on the specific requirements of the simulation, may need to include additional parameters in the URDF file. These parameters can include sensor definitions, controller configurations, or other custom elements related to the robot's behavior.

- **Save and Use the URDF File**: Save the URDF file with an appropriate filename and extension (e.g., seagate_robot.urdf). This file can now be used within the ROS ecosystem, such as with the Gazebo simulator or other ROS-based tools, to visualize and simulate the Seagate Mobile Robot.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

2. **Implement ROS2 Functionality**:

  - Set up a ROS2 workspace and create a package for the Seagate mobile robot simulation.
  - Configure the package dependencies to include ROS2 Gazebo and any additional packages required for the robot's specific functionality.

  - **ROS2 Packages**:

    - **ros-foxy-desktop**: This package includes the core ROS2 system, libraries, tools, and visualization components.
    - **ros-foxy-gazebo-ros-pkgs**: Provides ROS2 interfaces and plugins for integration with Gazebo simulation.

  - **Gazebo Packages:**

    - **gazebo9**: Gazebo is a powerful 3D robot simulation environment. Install Gazebo version 9, which is compatible with ROS2 Foxy.
    - **ros-foxy-gazebo-dev**: Provides additional development headers and libraries for Gazebo integration with ROS2.

  - **Robot Modeling Packages**:

    - **ros-foxy-robot-state-publisher**: Publishes the robot's state (e.g., joint positions) to the ROS2 system.
    - **ros-foxy-xacro**: Allows for the use of Xacro files, which are XML macros used for robot description.

- **Navigation Stack Packages**:

  - **ros-foxy-navigation2**: The ROS2 navigation stack, which provides mapping, path planning, and localization capabilities.
  - **ros-foxy-nav2-bringup**: Provides launch files and configuration for running the ROS2 navigation stack.

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content, and cite the document when use.

- **Sensor Simulation Packages**:

    - **ros-foxy-gazebo-plugins**: Includes Gazebo plugins for simulating sensors like LIDAR and cameras.
    - **ros-foxy-image-transport-plugins**: Enables image transport between Gazebo and ROS2.

- **Controller Packages**:

    - **ros-foxy-joint-state-controller**: Manages the joint states (e.g., positions, velocities) of the robot.
    - **ros-foxy-effort-controllers**: Provides controllers for managing the effort applied to the robot's joints.

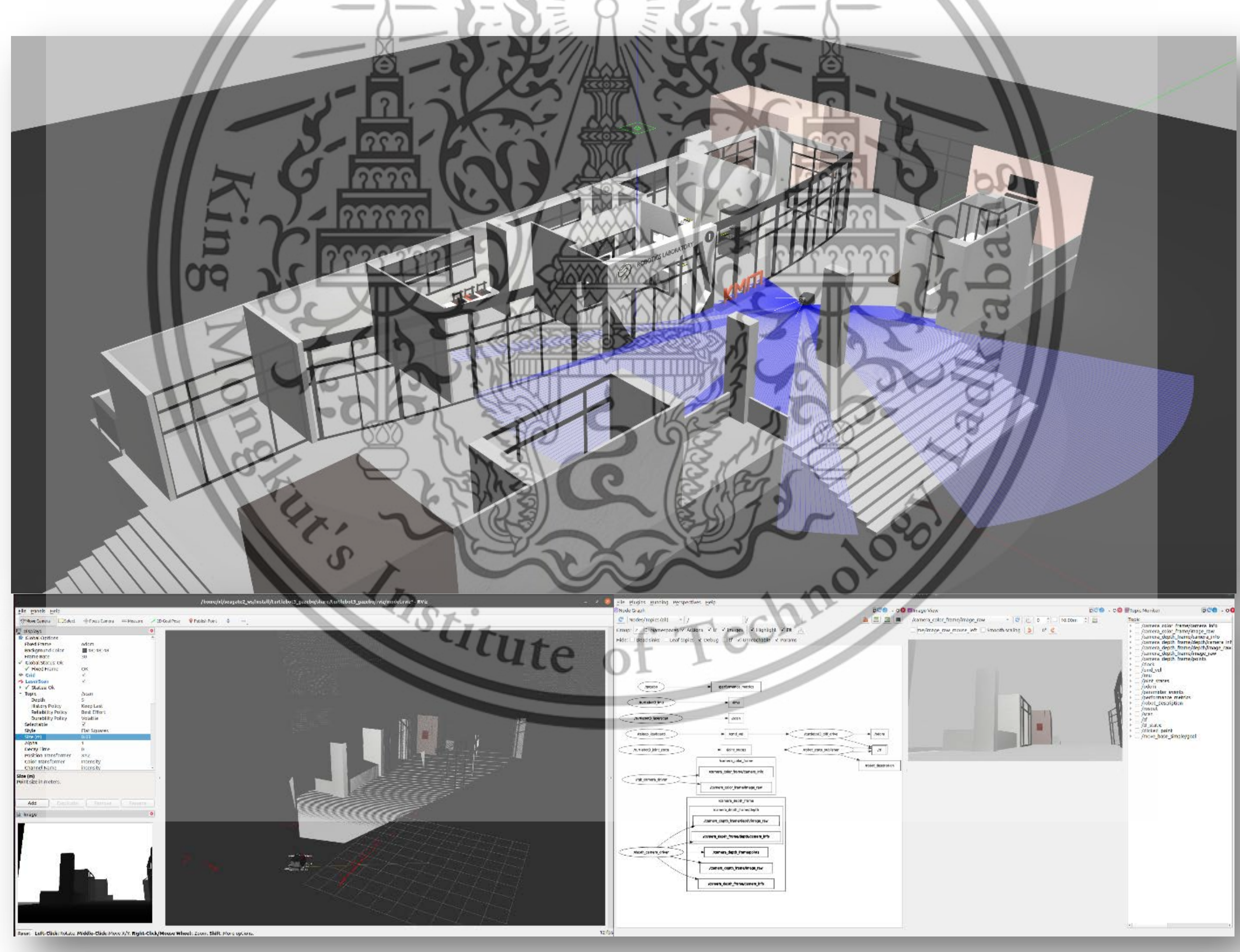


Figure 3. 16 Communication with other nodes during run a simulation

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

3. Seagate Robot simulation in Gazebo:

- Create a Gazebo launch file to spawn the Seagate mobile robot in the simulation environment. SDF (Simulation Description Format) file. The snippet includes a model named "turtlebot3_seagate2" and sets its initial pose in the simulation environment.

```
    <include>
      <uri>model://turtlebot3_seagate2</uri>
      <pose>-3.18 16.77 1.7 0 0 0</pose>
    </include>

    <include>
      <uri>model://turtlebot3_hm1st</uri>
      <pose>0 0 0 0 0 0</pose>
    </include>
```

  - **<include>**: This element is used to include another model within the current SDF file.
  - **<uri>**: Element specifies the path or identifier of the model to be included. In this case, it uses the model:// URI scheme followed by the model's name, "turtlebot3_seagate2".
  - **<pose>**: element defines the initial position and orientation of the included model. The pose values are represented in the order x y z roll pitch yaw. In the example,

    the pose is set to

```
      <pose>-3.18 16.77 1.7 0 0 0</pose>
```

    with rotation set to

```
      <pose>0 0 0 0 0 0</pose>
```

- Configure the Gazebo environment to replicate the real world, such as lighting, physics properties, and terrain.

```
<!-- spawn common environment.................................... -->
    <include>
      <uri>model://ground_plane</uri>
      <pose>0 0 0 0 0 0</pose>
    </include>

    <include>
```

King Mongkut's Institute of Technology Ladkrabang
This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

```
    <uri>model://sun</uri>
  </include>

  <scene>
    <shadows>false</shadows>
  </scene>
```

In this case

- **<uri>**: element specifies the path or identifier of the model to be included, again using the model:// URI scheme which available by Gazebo.

- **Lighting**: Gazebo allows user to customize the lighting conditions in the simulated environment to mimic real-world scenarios. adjusting the intensity, direction, and color of light sources to create realistic lighting effects. By replicating the lighting conditions of the real world

```
<!-- light source......................................... -->
  <light name='user_point_light_0' type='point'>
    <pose>0 0 10 0 0 0</pose>
    <diffuse>0.5 0.5 0.5 1</diffuse>
    <specular>0.1 0.1 0.1 1</specular>
    <attenuation>
      <range>20</range>
      <constant>0.5</constant>
      <linear>0.01</linear>
      <quadratic>0.001</quadratic>
    </attenuation>
    <cast_shadows>0</cast_shadows>
    <direction>0 0 -1</direction>
  </light>
```

- **Camera**: Basic configuration for a camera in Gazebo. which can customize the camera further by adjusting its pose, field of view, resolution, or other parameters based on your specific requirements. Additionally, it can add camera sensor plugins to generate image data from the camera and incorporate it into your simulation for perception tasks or computer vision algorithms.

King Mongkut's Institute of Technology Ladkrabang

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content and cite the document when use.

```xml
<!-- ......camera view .................................................
-->
    <gui fullscreen='0'>
      <camera name='user_camera'>
        <pose frame=''>-3.18 16.77 10 0 1.5138 0.009599</pose>
        <view_controller>orbit</view_controller>
        <projection_type>perspective</projection_type>
      </camera>
    </gui>

    <physics type="ode">
      <real_time_update_rate>1000.0</real_time_update_rate>
      <max_step_size>0.001</max_step_size>
      <real_time_factor>1</real_time_factor>
      <ode>
        <solver>
          <type>quick</type>
          <iters>150</iters>
          <precon_iters>0</precon_iters>
          <sor>1.400000</sor>
          <use_dynamic_moi_rescaling>1</use_dynamic_moi_rescaling>
        </solver>
        <constraints>
          <cfm>0.00001</cfm>
          <erp>0.2</erp>
          <contact_max_correcting_vel>2000.000000</contact_max_correcting_vel>
          <contact_surface_layer>0.01000</contact_surface_layer>
        </constraints>
      </ode>
    </physics>
```

- **Model**: Add various models to the Gazebo environment, such as robots, objects, or additional components, to create a realistic and dynamic simulation scenario.

```xml
<!-- spawn Seagate model ,
world.................................................. -->
    <include>
      <uri>model://turtlebot3_seagate2</uri>
      <pose>-3.18 16.77 1.7 0 0 0</pose>
    </include>

    <include>
      <uri>model://turtlebot3_hm1st</uri>
```

King Mongkut's Institute of Technology Ladkrabang
This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

```
    <pose>0 0 0 0 0 0</pose>
  </include>
```

- Include Gazebo plugins for sensors, such as a lidar plugin for generating simulated lidar data and camera plugins for simulating vision-based perception.

  - **IMU sensor plugin**
    The **libgazebo_ros_imu_sensor.so** library provides the necessary code to interface with the IMU sensor model in Gazebo and publish the sensor data as ROS messages. It contains the logic to generate simulated IMU data based on the configured noise parameters and the robot's dynamics in the simulation.

```
    <plugin name="turtlebot3_imu" filename="libgazebo_ros_imu_sensor.so">
      <ros>
        <!-- <namespace>/tb3</namespace> -->
        <remapping>~/out:=imu</remapping>
      </ros>
    </plugin>
```

  - <**plugin**>: Element is used to attach a Gazebo plugin to the robot's sensor. In this case, the plugin is named "turtlebot3_imu" and its corresponding library file is specified in the filename attribute.
  - <**ros**>: This section specifies the ROS-related configuration for the plugin.
  - <**remapping**>: Element is used to remap the output topic of the IMU sensor. In this example, the output topic is remapped from ~/out to imu. The ~ symbol represents the namespace of the robot, which can be customized.

  - **Lidar sensor plugin**
    The **libgazebo_ros_ray_sensor.so library** provides the necessary code to interface with the ray-based sensor model in Gazebo and publish the sensor data as ROS messages. The library handles the simulation of the sensor's behavior, including ray casting, detection of objects or obstacles, and the generation of simulated laser scan data.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

```
        <plugin name="turtlebot3_laserscan" 
filename="libgazebo_ros_ray_sensor.so">
          <ros>
            <!-- <namespace>/tb3</namespace> -->
            <remapping>~/out:=scan</remapping>
          </ros>
          <output_type>sensor_msgs/LaserScan</output_type>
          <frame_name>base_scan</frame_name>

        </plugin
```

Figure 3. 17 Seagate mobile robot laser scan range

- **3D camera**

  The **libgazebo_ros_camera.so** library provides the necessary code to interface with the camera sensor model in Gazebo and generate camera images based on the virtual scene. The library handles the simulation of the camera's behavior, including capturing images from a specified viewpoint, simulating perspective projection, and providing functionalities like adjusting image resolution, field of view, image format, and frame rate. plugin

```
        <plugin name="depth_camera_driver" filename="libgazebo_ros_camera.so">
          <ros>
          </ros>
          <update_rate>30</update_rate>
          <camera_name>camera_depth_frame</camera_name>
```

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

```
        <frame_name>camera_depth_optical_frame</frame_name>
        <hack_baseline>0.07</hack_baseline>
        <min_depth>0.001</min_depth>
      </plugin>
```

- **2D camera plugin**

  The *"**gazebo_ros_camera**"* plugin is part of the Gazebo ROS package and provides the necessary functionality to simulate a camera sensor in Gazebo and interface it with ROS. It allows you to configure and control the camera sensor properties such as image resolution, field of view, image format, frame rate, and camera pose.

```
      <plugin name="rgb_camera_driver" filename="libgazebo_ros_camera.so">
        <ros>
        </ros>
        <camera_name>camera_color_frame</camera_name>
        <frame_name>camera_color_optical_frame</frame_name>
        <hack_baseline>0.07</hack_baseline>
      </plugin>
```

- **Motor plugin**

  The "**libgazebo_ros_joint_state_publisher.so**" plugin is a Gazebo plugin that allows you to publish the joint states of your robot model in Gazebo to the ROS environment. It is part of the Gazebo ROS package and provides a bridge between Gazebo and ROS for joint state information.

```
    <plugin name="turtlebot3_joint_state"
filename="libgazebo_ros_joint_state_publisher.so">
      <ros>
        <remapping>~/out:=joint_states</remapping>
      </ros>
      <update_rate>30</update_rate>
      <joint_name>wheel_left_joint</joint_name>
      <joint_name>wheel_right_joint</joint_name>
    </plugin>
```

- Load the URDF model of the Seagate robot into Gazebo using the Gazebo launch file.
- Set initial positions and orientations for the robot within the simulation environment.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

4. **Implement Robot Control**:

- Develop the necessary ROS2 controllers for controlling the Seagate mobile robot's motor drive system, including velocity control and odometry calculation.
- Connect the controller nodes to the robot's simulated actuators (e.g., wheels) and sensors for receiving feedback.

5. **Map Generation and Navigation**:

- Implement ROS2 navigation stack (e.g., ROS2 Nav2) for path planning, mapping, and localization.
- Configure the navigation stack to utilize the simulated lidar sensor data for mapping the environment and generating the robot's trajectory.
- Set up navigation parameters, such as cost maps and motion planners, to enable the robot to navigate the simulated environment.

6. **Testing and Validation**:

- Run the Seagate mobile robot simulation in ROS2 Gazebo and test its behavior under various scenarios and conditions.
- Validate the accuracy of the simulated sensor outputs, such as lidar data and camera perception, by comparing them with the expected results based on the simulated environment.
- Evaluate the performance of the robot's control algorithms, navigation capabilities, and overall behavior.

7. **Iterative Refinement**:

- Analyze the simulation results and fine-tune the robot's control parameters, navigation algorithms, or sensor configurations as needed.
- Iterate the testing and validation process to continually improve the simulation's fidelity and the robot's performance.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

3.3 Migration

The migration process of ROS package from ROS1 to ROS2 can be broken down into a series of small steps. Each step will be explained in this section.

3.3.1 CMakeLists.txt: The first step is reviewing the dependencies and finding their direct equivalent in CMakeLists.txt and package.xml, because these are the index of the package, it specifies which dependencies used in building and compiling the package.

- **CMake version requirement:**
  - **ROS1:** ROS 1 uses CMake as its build system, and the minimum required version of CMake for ROS 1 is CMake 2.8.

```
cmake_minimum_required(VERSION 2.8.3)
project(package_name)
```

  - **ROS2:** In ROS2 the minimum version required should be 3.5 because ROS 2 leverages newer features and functionalities provided by CMake 3.x versions, which were not available in CMake 2.x.

```
cmake_minimum_required(VERSION 3.5)
project(package_name)
```

- **Find Package:** The **find_package()** function is used to locate and import dependencies, such as ROS packages or other libraries, into the project.
  - **ROS1:** In ROS1 the **find_package()** declaration follows this pattern, "**find_package(catkin REQUIRED COMPONENTS <package1> <package2> ...)**"

```
find_package(catkin REQUIRED COMPONENTS
  geometry_msgs
  roscpp
  std_msgs
)
```

  - **ROS2:** In ROS2 the **find_package()** declaration follows this pattern, "**find_package(<package1> REQUIRED)**"

```
find_package(ament_cmake REQUIRED)
```

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

```
find_package(geometry_msgs REQUIRED)
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)
```

- **Include Directories:** The **include_directories()** and **target_include_directories()** functions are used to specify the include directories for compiling source code.
  - **ROS1:** In ROS1 **${catkin_INCLUDE_DIRS}** includes the include directories of all the catkin dependencies of the package.

```
include_directories(include
  ${catkin_INCLUDE_DIRS}
)
```

  - **ROS2:** In ROS2 for more customizable and control over include directories, the **target_include_directories()** function is used to specify the include directories for the specific target.

```
target_include_directories(library_or_executable_target_name
  PUBLIC
    $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
    $<INSTALL_INTERFACE:include>
    ${ament_INCLUDE_DIRS}
    ${geometry_msgs_INCLUDE_DIRS}
    ${rclcpp_INCLUDE_DIRS}
    ${std_msgs_INCLUDE_DIRS}
)
```

- **Link Libraries:** The **target_link_libraries()** function is used to specify the libraries that a target (executable or library) depends on.
  - **ROS1:** In ROS1 the **${catkin_LIBRARIES}** includes the libraries of all the catkin dependencies of the package.

```
target_link_libraries(library_or_executable_target_name
  ${catkin_LIBRARIES}
)
```

  - **ROS2:** In ROS2 for more control over the included libraries, the **target_include_libraries()** function is used to specify the specific included libraries for a specific target.

```
target_link_libraries(library_or_executable_target_name
  ${geometry_msgs_LIBRARIES}
  ${rclcpp_LIBRARIES}
  ${std_msgs_LIBRARIES}
```

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

```
  ${ament_LIBRARIES}
)
```

- **Package:** In ROS 1 and ROS 2, the **catkin_package()** and **ament_package()** functions are used to declare metadata and dependencies in the package.xml file.
  - **ROS1:** In ROS 1, the function is used in the CMakeLists.txt file to generate the package.xml file and provide information about the package, its dependencies, and other metadata. This is specific to ROS1 that uses Catkin as its build system.

```
catkin_package()
```

  - **ROS2:** ROS 2 introduced a new build system called Ament and by using "**ament_package()**", ROS 2 packages can benefit from features such as better tracking of package dependencies, simplified package management, and enhanced integration with tools like "**colcon**" for building and managing the ROS 2 workspace.

```
ament_package()
```

3.3.2 Package.xml:

- **Format:**
  - **ROS1:** In ROS1 the system uses format 1 and 2, in format 1 it does not explicitly define dependencies on other packages, while format 2 does, and the default format for ROS1 is 2.

```
<?xml version="1.0"?>
<package format="1" > #or format = "2"
```

  - **ROS2:** Format 3 was introduced in ROS2, it provides more comprehensive metadata information about the package. Format 3 is used by build tools like "**colcon**" and "**ament**" to manage package dependencies, build, and installation.

```
<?xml version="1.0"?>
<package format="3"> #or format = "2"
```

- **Run/Execute dependencies**
  - **ROS1:** The **run_depend** tag is used for libraries, messages, and other resources that are necessary for the package to operate at runtime.

```
  <run_depend>geometry_msgs</run_depend>
  <run_depend>roscpp</run_depend>
```

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

```
  <run_depend>std_msgs</run_depend>
```

- **ROS2:** The **exec_depend** tag is used for the same purpose as **run_depend** in ROS1 for the exception that it can also cover executable.

```
  <exec_depend>geometry_msgs</exec_depend>
  <exec_depend>rclcpp</exec_depend>
  <exec_depend>std_msgs</exec_depend>
```

3.3.3 Main source code:

- **Header**
  - **ROS1:** In ROS1 the common extension for the header is .h files.

```
#include <ros/ros.h>
#include <std_msgs/Float32.h>
#include <geometry_msgs/Twist.h>
#include <geometry_msgs/TwistWithCovarianceStamped.h>
```

- **ROS2:** In ROS2, the preferred extension for the header file is .hpp, although it is still possible to use .h but, it is not common to do so. To migrate to ROS2, the common practice for the header is either having a direct equivalent or similar packages included in its place. As for the package structure, there are some changes. For example, in ROS1, "**message_package/header.h**" becomes "**message_package/msg/header.hpp**" in ROS2.

```
#include <rclcpp/rclcpp.hpp>
#include <std_msgs/msg/float32.hpp>
#include <geometry_msgs/msg/twist.hpp>
#include <geometry_msgs/msg/twist_with_covariance_stamped.hpp>
```

- **Variable Declaration**
  - **ROS1:** Variable declaration in ROS1 uses the structure of “**datatype variable_name;**”. The example is shown below, The variable name “**message_var**” declared as the datatype of “**geometry_msgs/Twist**”.

```
geometry_msgs::Twist message_var;
```

- **ROS2:** The same as in ROS1 but with minor changes to the data structures for the message/action/service datatype.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

```
geometry_msgs::msg::Twist message_var;
```

- **Function calls:**
  - **ROS1:** The variable "**var_time**" is defined to inherit the returned value from the function "**ros::Time::Now**" in the "**ros::Time**" class.

```
var_time = ros::Time::now();
```

  - **ROS2:** There is no direct equivalent of the function available, the changes to the function that has similar output is necessary.

```
var_time = rclcpp::Clock().now();
```

- **Parameter Declaration**
  - **ROS1:** In ROS1 parameters can be directly assigned in one line.

```
double var_param_name;
node.param<double>("param_name", var_param_name, 0.24);
```

  - **ROS2:** In ROS2 parameters need to be assigned to the inherited pointers of node, and then assign the value to the variable after.

```
double var_param_name;
this->declare_parameter<double>("param_name", 0.24);
var_param_name = this->get_parameter("param_name").as_double();
```

- **Publisher Declaration**
  - **ROS1:** In ROS1, there is no need to mention the datatype of the message in the variable declaration, only in the advertise line, and 'node' is a "**ros::NodeHandle**" object representing the node.

```
ros::Publisher publisher_var;
publisher_var = node.advertise<geometry_msgs::Twist>("topic_name", 100);
publisher_var->publish(message);
```

  - **ROS2:** In ROS2, in the variable declaration there is a need to declare the message type and make it a **SharedPtr** type. In the "**create_publisher**" function "**node**" is a "**rclcpp::Node::SharedPtr**" object representing the node.

```
rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr publisher_var;
```

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

```
publisher_var = node-
>create_publisher<geometry_msgs::msg::Twist>("topic_name", 100);
publisher_var.publish(message);
```

- **Subscriber Declaration**
  - **ROS1:** In ROS1, there is no need to mention the datatype of the message in the variable declaration, only in the subscribe line, and 'node' is a "**ros::NodeHandle**" object representing the node.

```
ros::Subscriber subscriber_var;
subscriber_var = node.subscribe("topic_name", 10,
&SubscribeAndPublish::Function_Callback, this);
```

  - **ROS2:** In ROS2, in the variable declaration there is a need to declare the message type and make it a **SharedPtr** type. In the **create_subscriber** function "**Node**" is a "**rclcpp::Node::SharedPtr**" object representing the node.

```
rclcpp::Subscription<std_msgs::msg::Float32>::SharedPtr subscriber_var;
subscriber_var = node->create_subscription<std_msgs::msg::Float32>("rpm", 10,
std::bind(&SubscribeAndPublish::Function_Callback, this,
std::placeholders::_1));
```

- **Main (spin node)**
  - **ROS1:** In ROS1, "**ros::init()**" also needs to declare the node name, and need to call the object/function seperately from the node spin.

```
int main(int argc, char** argv){

  ros::init(argc, argv, "node_name");
  SubscribeAndPublish Object;
  ros::spin();
}
```

  - **ROS2:** In ROS2, "**rclcpp::init**" has no need to declare the node name, because the node name will be declared in the node declaration class, for "**rclcpp::spin()**" this can now call the function together with the node. Additionally, in ROS2 there is a need to call "**rclcpp::shutdown()**" to shutdown the node completely when the process is done.

```
int main(int argc, char** argv){
```

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

```
    rclcpp::init(argc, argv);
    rclcpp::spin(std::make_shared<SubscribeAndPublish>());
    rclcpp::shutdown();
    return 0;
}
```

3.3.4 Launch file (.xml)

- **ROS1**: ROS1 uses only XML file to write launch files.

```
<?xml version="1.0"?>

<launch>

  <node pkg="package_name" type="executable_name" name="source_cpp_name"
      output="screen" respawn="true">
</launch>
```

- **ROS2:** ROS2 launch file is mainly written in Python although XML launch file is still usable.

```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        Node(
            package="package_name",
            executable="executable_name",
            name="source_cpp_name",
            output="screen",
            respawn=True,
            parameters=[
                {"param_name_1": "param_1"},
                {"param_name_2": "param_2"},
                {"param_name_3": "param_3"},
                {"param_name_4": True},
                {"param_name_5": 0.5}
            ]
        )
    ])
```

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

CHAPTER 4

EXPERIMENTAL RESULT

4.1 Simulation:

The Seagate mobile robot, deployed for educational purposes and testing in simulation, has shown promising results. The robot has successfully demonstrated its capabilities in various tasks and scenarios, showcasing its potential as a versatile educational platform and a reliable testing tool.

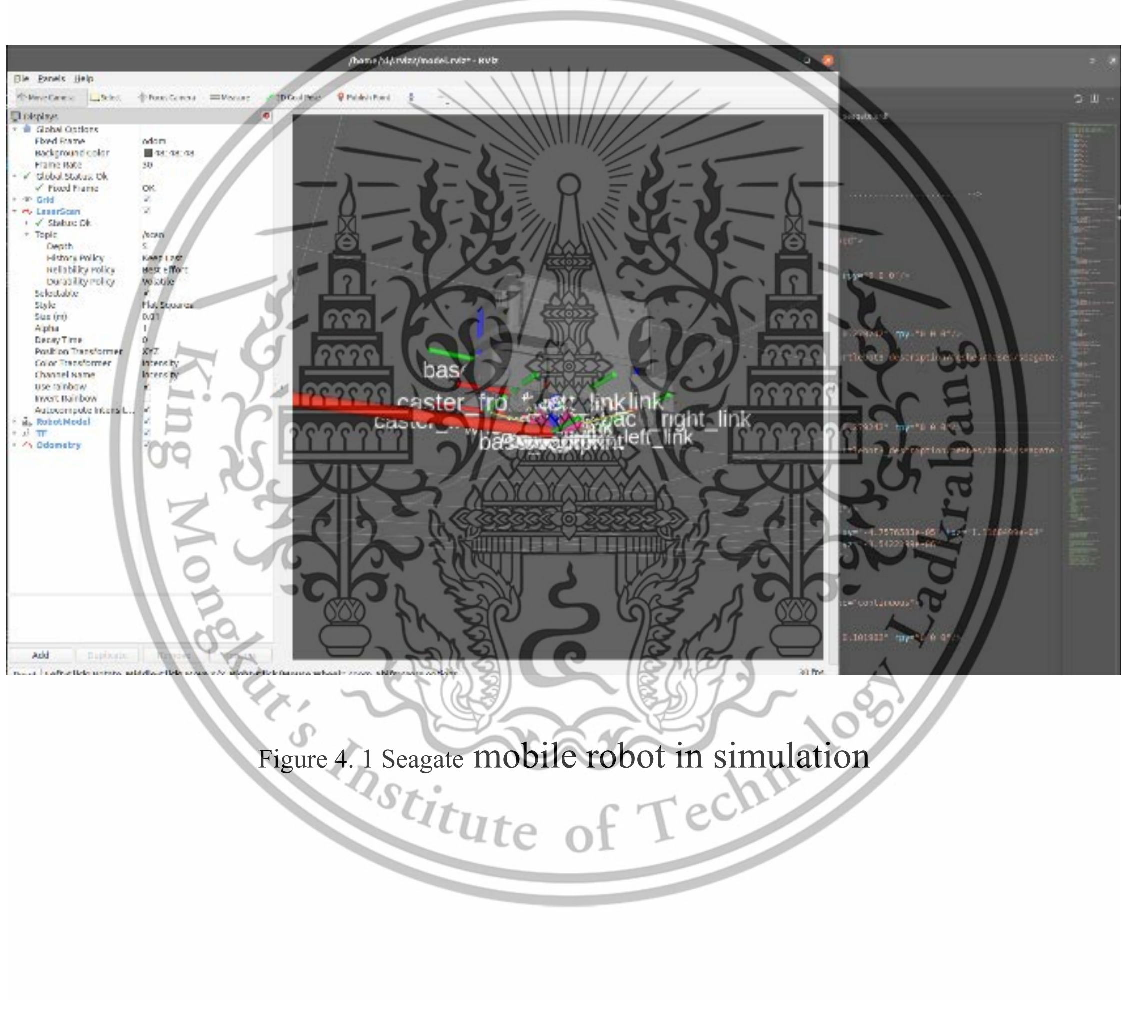


Figure 4. 1 Seagate mobile robot in simulation

King Mongkut's Institute of Technology Ladkrabang

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

4.1.1 navigation and path planning: The Seagate mobile robot has been able to autonomously navigate through complex environments, avoiding obstacles and following planned paths with a high degree of accuracy. The robot's robustness in handling different map configurations and dynamic obstacles has been demonstrated through extensive testing.

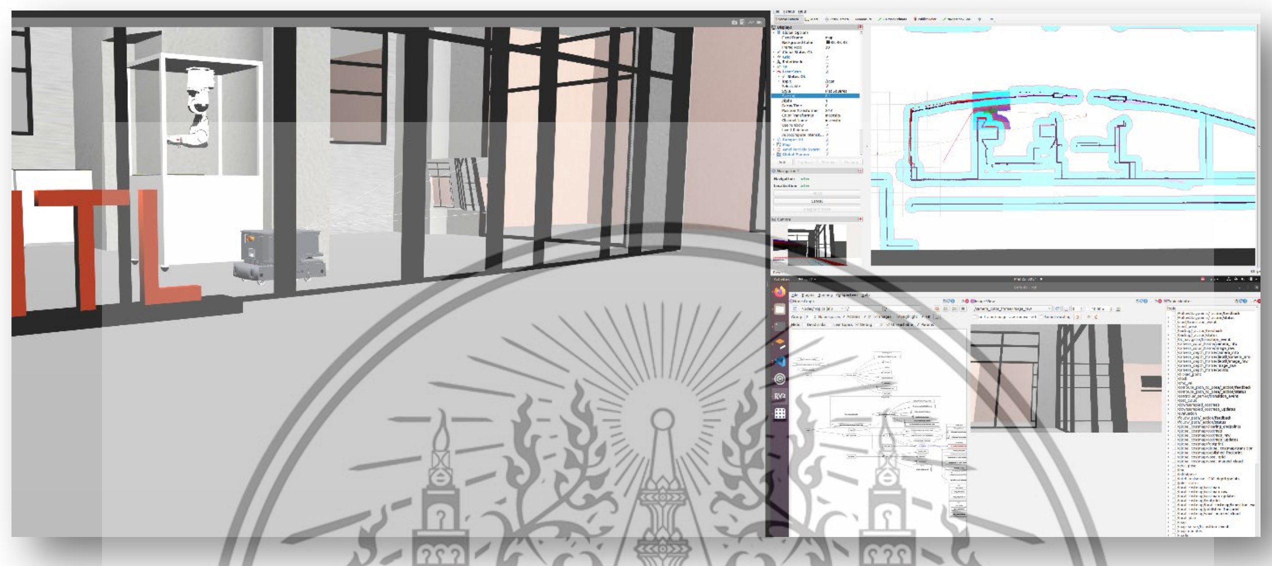


Figure 4. 2 Seagate mobile robot simulation running navigation NAV2

4.1.2 Robot sensing: Including lidar, cameras, and IMU, has proven effective in providing accurate perception and environment understanding. The sensors have enabled the robot to accurately detect and localize objects in its surroundings, facilitating obstacle avoidance and decision-making during navigation.

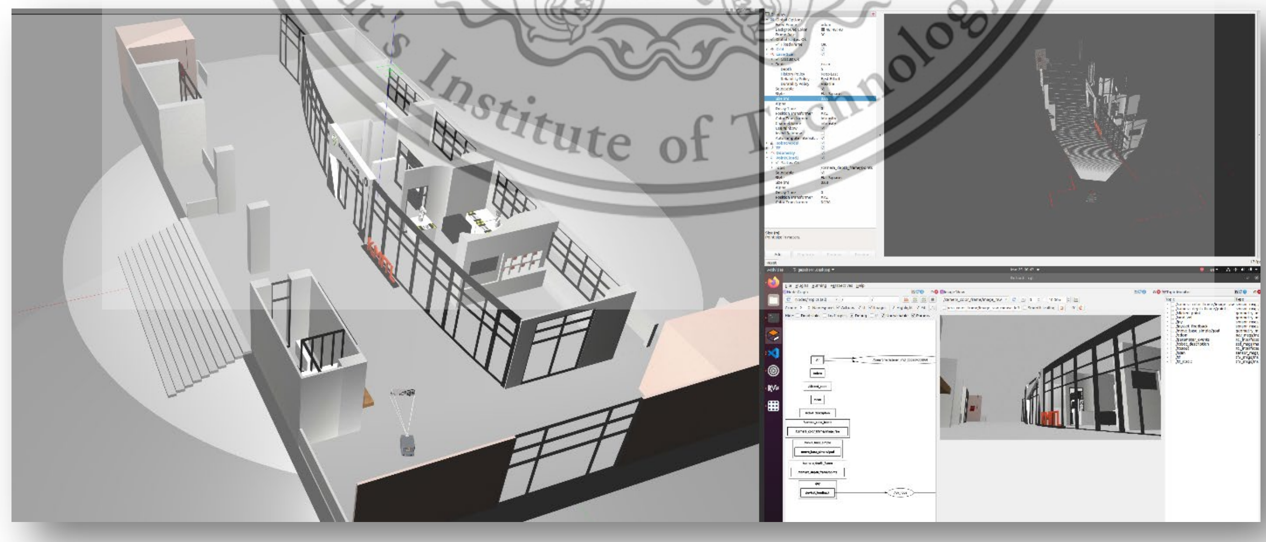


Figure 4. 3 Visualize the output topic by rviz ,rqt

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

King Mongkut's Institute of Technology Ladkrabang

4.1.3 Teleoperation capability: The Seagate mobile robot has been successfully tested, allowing users to remotely control and maneuver the robot using intuitive interfaces. The robot's response to teleoperation commands has been prompt and reliable, ensuring smooth remote-control operation.

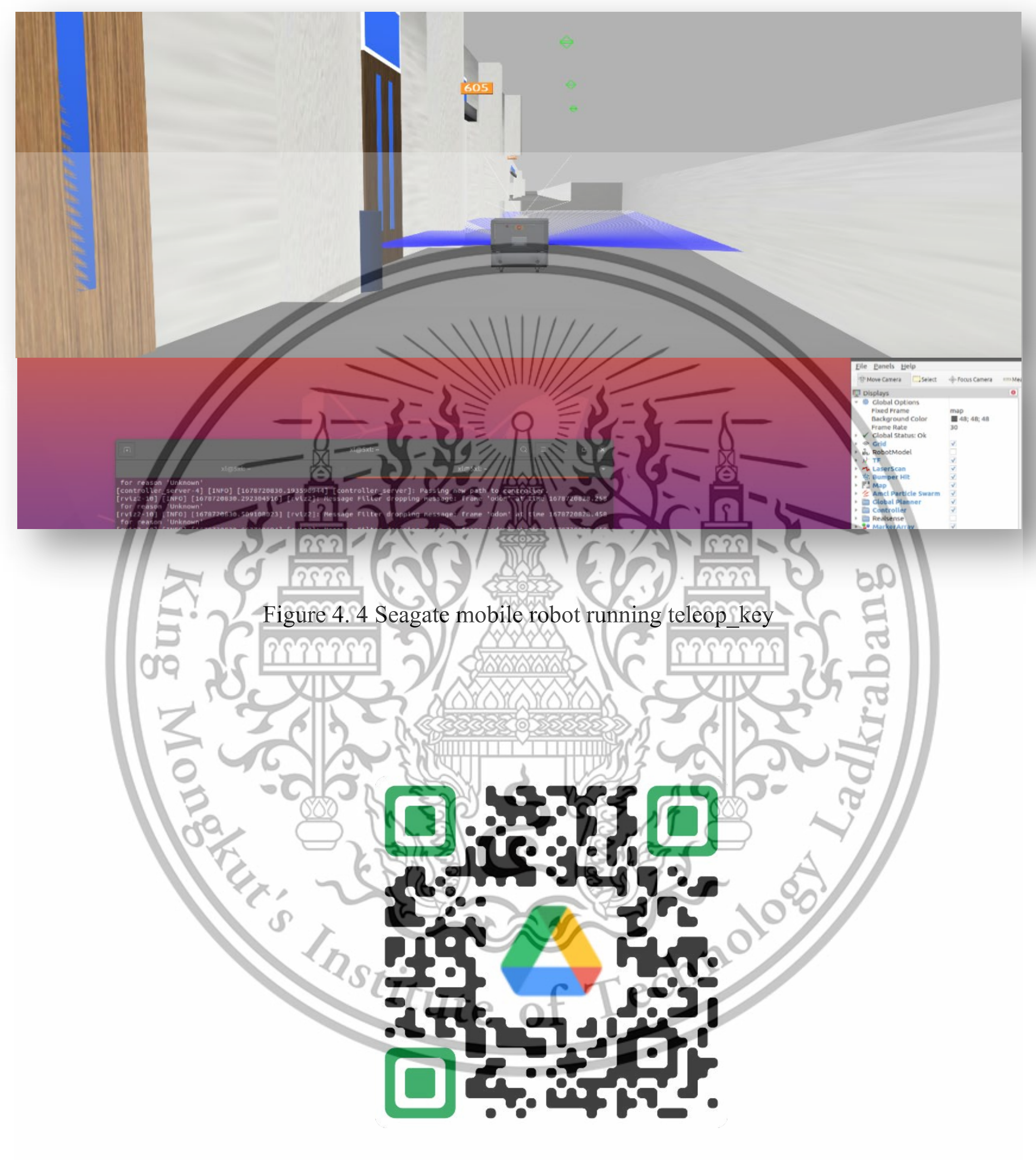


Figure 4. 4 Seagate mobile robot running teleop_key

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

4.1.4 3D maps benefits:

For the part of creating 3D maps.3D maps benefit robot simulations by providing realistic environments, simulating sensor data, aiding path planning and navigation, enabling object recognition and manipulation training, allowing for simulation of dynamic environments, and facilitating safe and cost-effective training and testing of robot systems. They enhance the accuracy and realism of simulations, helping robots navigate and interact with their surroundings as they would in the real world.

King Mongkut's Institute of Technology Ladkrabang

Figure 4. 5 Final of 3D HM Robotics Laboratory map

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

Figure 4. 6 Final view of 3D HM 6th floor map

Figure 4. 7 Final view of E12 12th floor map

King Mongkut's Institute of Technology Ladkrabang
This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

4.2 Migration: All ROS1 packages have been successfully migrated to ROS2 Foxy and can be used as a base package for further development in the next ROS2 distro releases.

4.2.1 Topic Monitoring: Topic Monitoring is a method used to view all the running nodes. Topic monitoring also provides all publishing and subscribing topics in each node.

- **ROS1**: The "**rqt_graph**" is a tool providing GUI for topic monitoring.

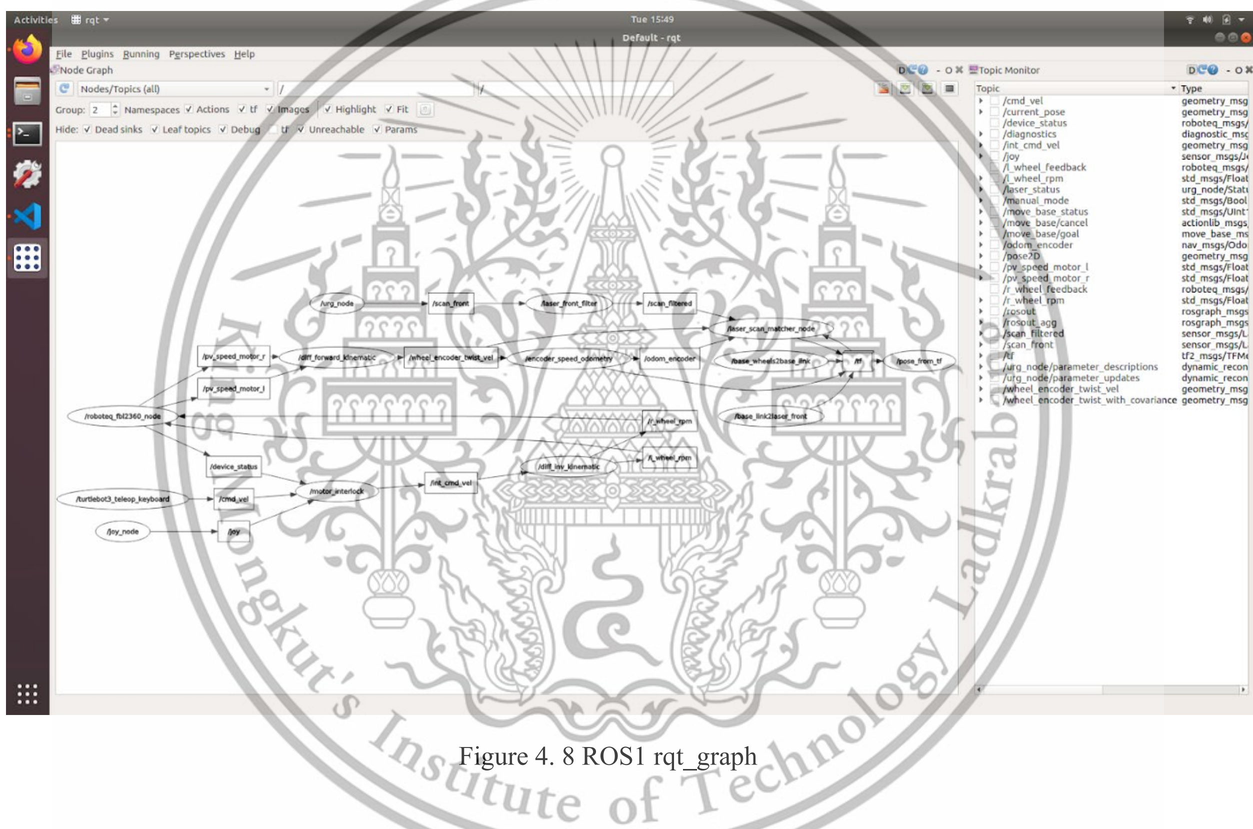


Figure 4. 8 ROS1 rqt_graph

- **ROS2**: For ROS2, "**rqt_graph**" also can be used to perform topic monitoring.

King Mongkut's Institute of Technology Ladkrabang
This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

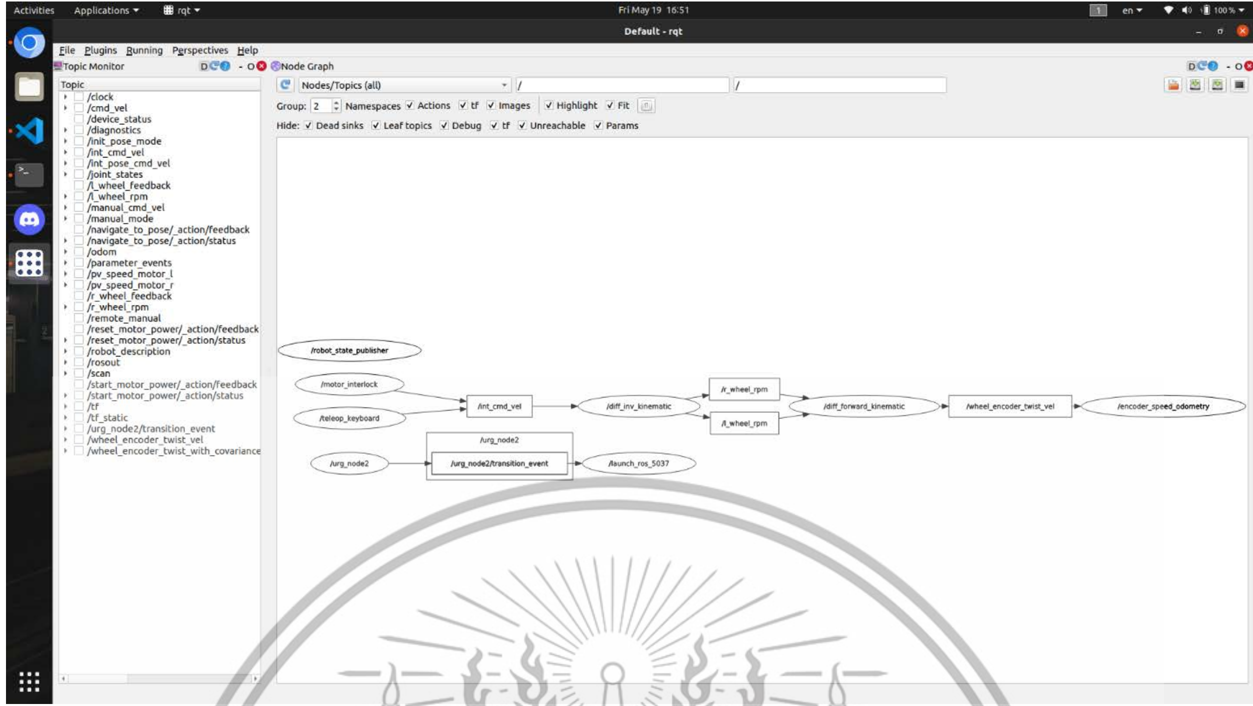


Figure 4. 9 ROS2 rqt_graph

One problem to mention is ROS2 does not have all packages from ROS1 migrated. From the Figure (ROS2), one node called "**laser_scan_matcher"** is missing in Fig (ROS2). Now, ROS2 Foxy still does not have the equivalent package for "**laser_scan_matcher**".

4.2.2 Visualization: "**rviz2**" is the method for developers to monitor the current state, pose, sensor condition of the robot. The result of SEAGATE robot package migration to ROS2 is a success. All data from the Lidar sensor, the pose, the motor driver, and the odometry is obtainable.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

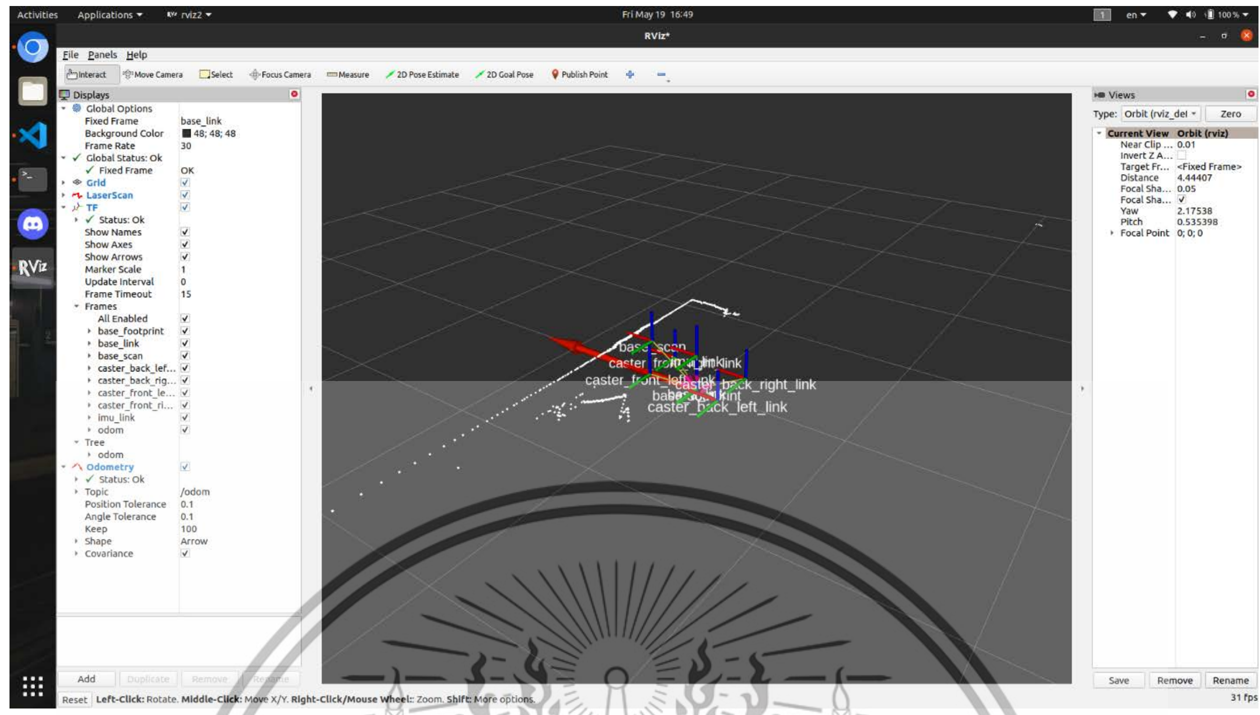


Figure 4. 10 Visualization via ROS2 RVIZ

4.2.3 SLAM: ROS2 utilizes the "**cartographer**" method for mapping. The ROS1 version of the SEAGATE robot used the "**hector_mapping**" method. So, the challenge for the mapping for the ROS2 version is learning how to use "**cartographer**". One encountered problem while testing the SEAGATE robot in SLAM is the unstable motion. The cause of the issue might come from robot transportation. Scratches from friction or accident may have an effect in damaging the wheels. The issue has been one major obstacle of the project from experimenting with navigation.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

Figure 4. 11 SLAM via cartographer

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

CHAPTER 5
CONCLUSION

The deployment of the Seagate mobile robot in simulation has allowed for the creation and evaluation of detailed 3D maps. These maps not only serve as a valuable tool for simulating and testing robot behavior but also have the potential for broader applications. The 3D map models can be further developed and adapted for various scenarios, such as augmented reality (AR) and virtual reality (VR) tours.

The map's compatibility with AR/VR technologies opens possibilities for creating immersive experiences and virtual tours in diverse environments, including historical sites, museums, or architectural walkthroughs. Users can explore and interact with the virtual representation of the real-world environment, leveraging the detailed 3D map model created during the Seagate mobile robot's simulation.

The migration from ROS1 to ROS2 is a crucial step in advancing the Robot Operating System. This thesis has explored the migration process in detail, covering the necessary modifications to CMakeLists.txt, package.xml, and source code.

By following the steps outlined in this research, developers can seamlessly modify their ROS1 packages to ensure compatibility with ROS2. This smooth transition results in a ROS2 codebase that can be easily migrated to future distro versions, extending the package's maintenance lifespan.

Overall, the experimental results showcase the Seagate mobile robot's capabilities as an educational and testing platform in simulation and real-world. The robot's navigation, sensing, teleoperation, and map development functionalities provide a solid foundation for educational purposes and extend to broader applications in the field of virtual tours and immersive experiences.

King Mongkut's Institute of Technology Ladkrabang

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content, and cite the document when use.

REFERENCES

[1] Brush, K. (2019) *What is a mobile robot? definition from whatis.com.*, *IoT Agenda*. Available at: https://www.techtarget.com/iotagenda/definition/mobile-robot-mobile-robotics

[2] *What are industrial cleanrooms & how are they used? Thomasnet® - Product Sourcing and Supplier Discovery Platform - Find North American Manufacturers, Suppliers and Industrial Companies*. Available at: https://www.thomasnet.com/insights/what-are-industrial-cleanrooms-how-are-they-used-/

[3] *What is linux? Linux.com*. Available at: https://www.linux.com/what-is-linux/

[4] Abubakar, M. (2021) *What is ubuntu?* Available at: https://www.howtogeek.com/763775/what-is-ubuntu/

[5] *Why Ros? ROS*. Available at: https://www.ros.org/blog/why-ros/

[6] *What is Blender Used For*? Available at: https://www.blenderbasecamp.com/home/what-is-blender-used-for-a-list-of-reasons-to-use-blender/

[7] *What is Fusion 360?* Available at: https://www.autodesk.com/solutions/what-is-fusion-360

[8] *What Is an OBJ File and How Do You Use It?* Available at: https://www.makeuseof.com/what-is-an-obj-file/

[9] *What is a DAE file?* Available at: https://docs.fileformat.com/3d/dae/

[10] *Getting Started ROS*. Available at: https://www.ros.org/blog/getting-started/ (Accessed: 09 May 2023).

[11] Ed (2021) *ROS1 vs Ros2, practical overview for ROS developers - the robotics back*, *End*. Available at: https://roboticsbackend.com/ros1-vs-ros2-practical-overview/ (Accessed: 09 May 2023).

[12] Zandra B. (2019) Unmanned Ground Vehicle Modelling in Gazebo/ROS-Based Environments. Department of Industrial Engineering, University of Salerno.

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content and cite the document when use.

[13] Afanasyev, I.M., Sagitov, A.G., and Magid, E.A. 2016. ROS-based SLAM for a Gazebo-simulated mobile robot in image-based 3D model of indoor environment. LNCS, 9386.

[14] Shimchik, I., Sagitov, A., Afanasyev, I., Matsuno, F., and Magid, E. 2016. Golf cart prototype development and navigation simulation using ROS and Gazebo. Int. Conf. on Mechanical, System and Control Engineering, MATEC Web Conf., 75, 0900.

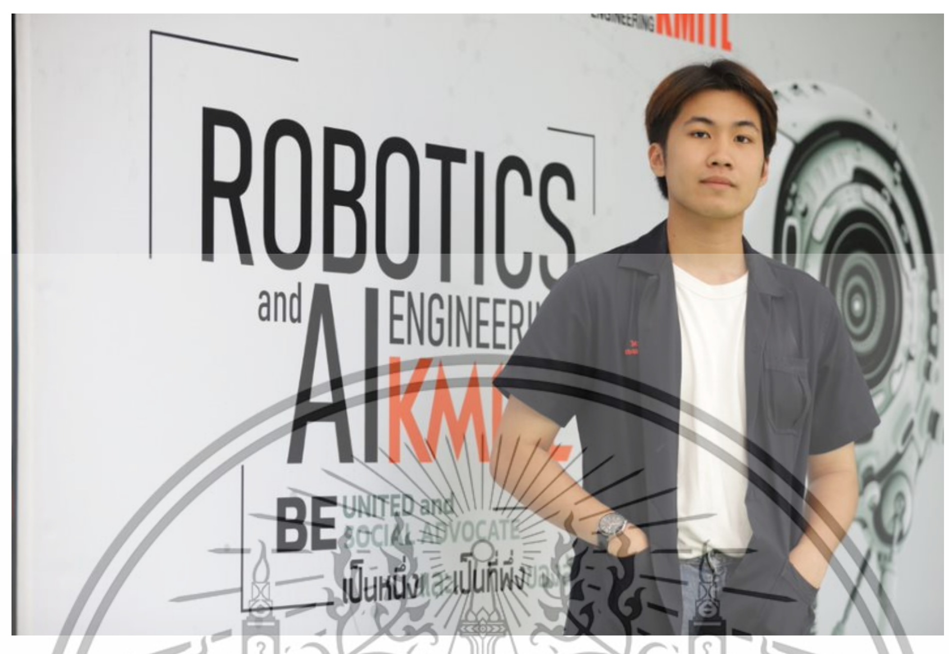
[15] *ROS Wiki ros.org*. Available at: http://wiki.ros.org/melodic

[16] *Ros2-how is it better than ROS1* (2022) *Medium*. Available at: https://medium.com/@oelmofty/ros2-how-is-it-better-than-ros1-881632e1979a

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content, and cite the document when use.

BIOGRAPHY



The project member Arin Madnurak is a dedicated student currently pursuing 4th year Robotics and AI Engineering at KMITL (King Mongkut's Institute of Technology Ladkrabang). With a passion for cutting-edge technology, Arin has been actively involved in numerous projects that showcase his expertise in the field.

One notable project that highlights Arin's skills is the development of the Seagate mobile robot. This project demonstrated his ability to design, build, and program a mobile robot capable of performing various tasks. Through this project, Arin gained valuable hands-on experience in robotics and honed his problem-solving abilities.

Arin's enthusiasm for learning and practical experience extends beyond the academic realm. He has completed internships at Singha Corporation and Wasabi Ventures, where he further refined his skills and gained insights into the professional world. These opportunities provided him with exposure to industry practices and a chance to collaborate with experts in the field.

With a strong foundation in robotics, AI, and a diverse range of experiences, Arin Madnurak continues to pursue his passion for innovation and problem-solving. As he progresses in his academic journey and ventures into the professional world, he strives to make a significant impact in the fields of robotics, AI, and beyond.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

BIOGRAPHY

Thanapat Pakornkittibown, born on November 14, 2000, is an ambitious student pursuing a degree in Robotics and AI Engineering at King Mongkut's Institute of Technology Ladkrabang (KMITL) in Bangkok, Thailand. Throughout his academic journey, he has actively participated in projects and competitions, showcasing his expertise and dedication.

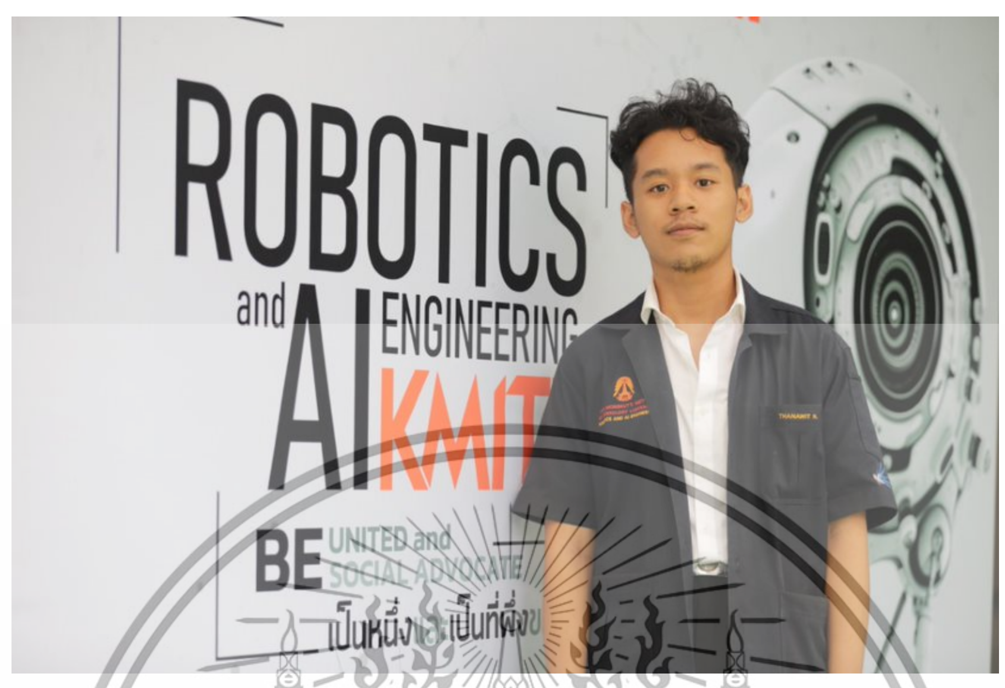
With a solid foundation in Robotics and AI Engineering, Thanapat completed a two-month internship at Hutchison Ports, contributing to the development of the Hutchison Ports OCR Portal System. He played a key role as a software developer, implementing a license plate detection and Optical Character Recognition (OCR) system, as well as developing machine vision technology for container information extraction.

Additionally, Thanapat gained valuable experience as a Subsea Robotics Engineering intern at AI & Robotics Ventures (ARV). During this six-month cooperative internship, he focused on Forward Looking Sonar simulation and ROS package migration, contributing to the advancement of cutting-edge robotics technologies.

Thanapat's achievements extend beyond his academic pursuits. He secured the 2nd Runner-up position in the AWS Build on ASEAN competition, demonstrating his ability in providing innovative solutions.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

BIOGRAPHY

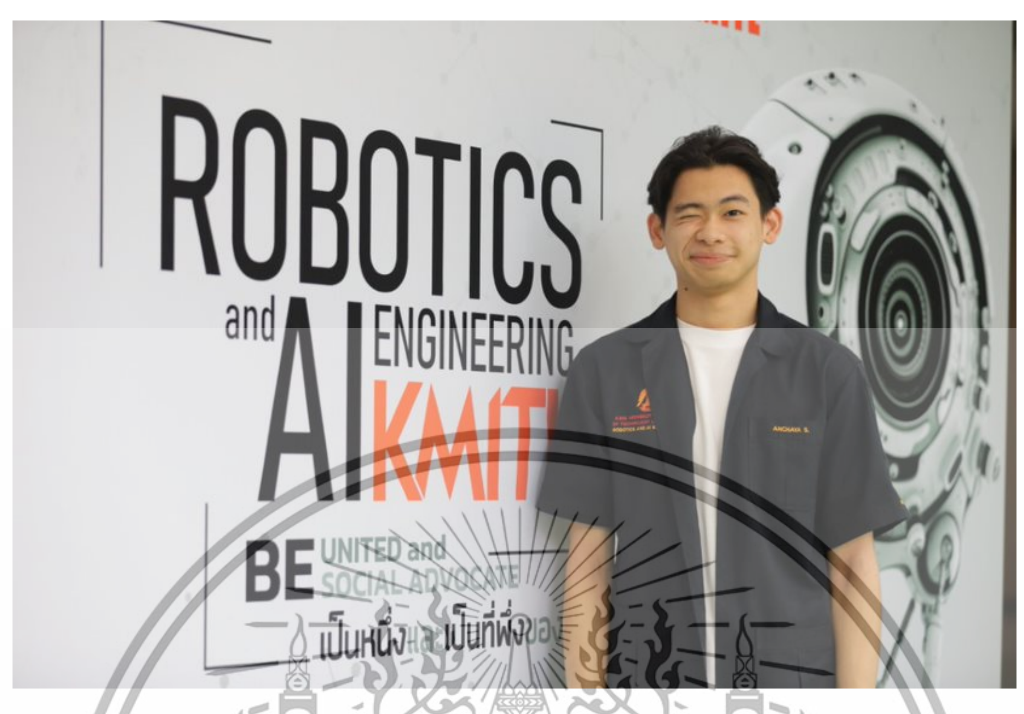


Thanawit Norkam, a fourth-year student at King Mongkut's Institute of Technology LadKrabang (KMITL), driven and ambitious individual who left his hometown of Chiang Mai, Thailand, to pursue his dreams in Robotics and AI Engineering. With a passion for science and technology from a young age, he joined KMITL. Fascinated by mobile robotics, Thanawit joined a project focused on the SEAGATE robot during his 4th year of university, aiming to implement his acquired knowledge to solve the industrial problems. His meticulous research and technical expertise have garnered attention and acclaim. In addition to his academic pursuits, he participates in robotics competitions, honing his skills in teamwork and problem-solving.

Thanawit's dedication and achievements have earned him accolades within the field of robotics. His inspiring journey from Chiang Mai to KMITL exemplifies his determination and intellectual curiosity, promising a future filled with groundbreaking advancements.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

BIOGRAPHY



The project member, Thitisart Thitathan was born on October 18th,2000, in Bangkok, Thailand. After all experiences and outcomes as an Robotics and AI engineering student at King Mongkut's Institute of Technology ladkrabang, Since the beginning of being a junior year1 student until now, senior year4 ,and in less than a week , I will be got a bachelor's degree, all this time, it's reminds me that a great engineer is built, not born. And it's the time where my idea drives me to step up into an environment where skilled engineers are groomed like nowhere else. I am certain that my skills and dedicated personality will be placed on the right path benefiting society in future someday.

Blog : Home | Thitiport (thitisart-mongkorn.wixsite.com)

20 may 2023, 3:09 am

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content and cite the document when use.

SEAGATE ROBOT MK. II (SIMULATION AND MIGRATION)

Arin Madnurak
Robotics and AI
Department of Engineering, KMITL

Thanawit Norkam
Robotics and AI
Department of Engineering, KMITL

Thitisart Thitathan
Robotics and AI
Department of Engineering, KMITL

Thanapat Pakornkittiboworn
Robotics and AI
Department of Engineering, KMITL

***Abstract*— Open-source software plays a pivotal role in the advancement of technology, particularly in the field of robotics. The continuous evolution of open-source software follows a distinct life cycle, comprising various stages such as development, testing, deployment, and maintenance. One such prominent open-source software platform is the Robot Operating System (ROS), which has revolutionized the development of robotics software.**

This report aims to explore the life cycle of open-source software, with a specific focus on the two major versions of ROS: ROS1 and ROS2. By understanding the nuances of these versions, educators in the field of mobile robotics can effectively adapt their teaching methodologies to incorporate the latest advancements in software development. Additionally, the report delves into the topic of robot simulation in Gazebo, a powerful simulation environment commonly used in conjunction with ROS.

Keywords—ROS, Gazebo, Migration, simulation, Mobile robot.

I. INTRODUCTION

Seagate is an innovative mobile robot designed for various applications such as warehouse automation, coordination, and indoor navigation. To ensure its efficient operation and performance, Algorithm test and validation is required before real-world deployment. Mobile robot simulation provides a valuable platform to accomplish this by creating virtual environments that closely mimic the robot's behavior and interactions with its surroundings. Furthermore, the Seagate robot's packages will be deprecated. The current framework used in Seagate robot is still ROS1 and Ubuntu 18.04 LTS. The End-of-Life state of any open-source software is inevitable. The End-of-Life period of Ubuntu 18.04 LTS is in April 2023. The result of End-of-Life software is no further development. The package migration is a crucial part for the Seagate mobile robot. With the upcoming trend of implementing ROS2 in robotics development project and the newly introduced features, Migration allows the developer to use the old packages but changing the structure of the source code instead of creating a new package for similar tasks.

II. CONCEPT, THEORIES, AND RELATED RESEARCH

A. *Mobile Robot:* A mobile robot [1] is a specific type of robot or machine controlled by software utilizing sensors or other technology (i.e., 2D,3D camera) to identify the covered surroundings and move around the environment. Mobile robots function using a combination of artificial intelligence (AI) and physical robotic elements, such as wheels, tracks, and legs. Mobile robots are becoming increasingly popular across different business sectors. Mobile robots are also used to help with work processes and even accomplish impossible or harmful tasks for humans.

Figure II.1 Seagate mobile robot 3D model

B. *Physics-based simulation*: Gazebo uses a physics engine to simulate the interactions between objects in the virtual environment, including dynamics such as gravity, friction, and collisions. Physics-based simulation allows for realistic simulations of robots and their environments.

C. *Sensor simulation*: Gazebo allows for the simulation of various sensors (such as cameras, lidars, and IMUs) which can be used to provide sensor data to a robot's control system. Sensor simulation allows for the testing of sensor-based navigation and perception algorithms.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

D. *Robot Operating System:* ROS [2] (Robot Operating System) is an open-source software development framework for robotics applications. ROS offers a standard software platform to developers across industries helping in research and prototyping all the way through to deployment and production.

E. *ROS Melodic Morenia*: ROS Melodic Morenia [3] is the twelfth ROS distribution released on May 23rd, 2018. ROS Melodic Morenia was created to fully function on Ubuntu 18.04 LTS despite having support in other operating systems such as Mac OS X and Window.

F. *ROS Foxy Fitzroy*: ROS Foxy Fitzroy [4] is the first LTS (Long Term Support) of ROS2 targeting Ubuntu 20.04 (Focal Fossa). ROS Foxy introduces new features such as security enhancements via Data Distribution Service (DDS), new methods of communication with the lack of ROS Master, increased performance regardless of the network situation, multi-platform support (Linux, Windows, macOS)

G. *Gazebo integration with ROS2*: Gazebo is a stand-alone application which can be used independently of ROS or ROS 2. The integration of Gazebo with either ROS version is done through a set of packages called "gazebo_ros_pkgs". These packages provide a bridge between Gazebo's C++ API and transport system, and ROS 2 messages and services.

H. *3D Format for Gazebo*: Gazebo can support various 3D file formats and there are many ways to import and visualize 3D models for Gazebo, enabling the creation of realistic virtual environments for robot simulations. By using these formats, the user can define the geometry, appearance, and physical properties of objects in the simulation, allowing for accurate and detailed simulations of robotic systems.

I. *Robot and Environment Modelling*: URDF file: URDF (Universal Robotic Description Format) is a file format used to describe the physical and kinematic properties of a robot in a standardized way. It is an XML file that contains information such as the links, joints, sensors, and visual representations of the robot. Procedures of describing a robot via URDF are as follows:

1.Define the links that make up the robot. Each link has a name and can have a visual and collision representation. The visual representation is what the robot looks like, while the collision representation is what other objects see when colliding with the robot.

2.define the joints that connect the links together. Joints describe how one link can move relative to another link, and they can be of various types such as revolute, prismatic, or fixed. Additionally, sensors and controllers can be added to the robot to control its movements and interactions with the environment.

SDF file: SDF (Simulation Description Format) file format is an XML-based file format used to describe models in Gazebo, the open-source robot simulation software. SDF files define the structure, properties, and behavior of entities within the simulation, including robots, objects, sensors, and environments. The process of conversion from URDF to SDF can be easily done by adding the plugins called **"gazebo_plugins"** into URDF file. The gazebo plugins can attach to ROS messages and service calls the sensor outputs and driving motor inputs, i.e., the gazebo plugins create a complete interface (Topic) between ROS and Gazebo.

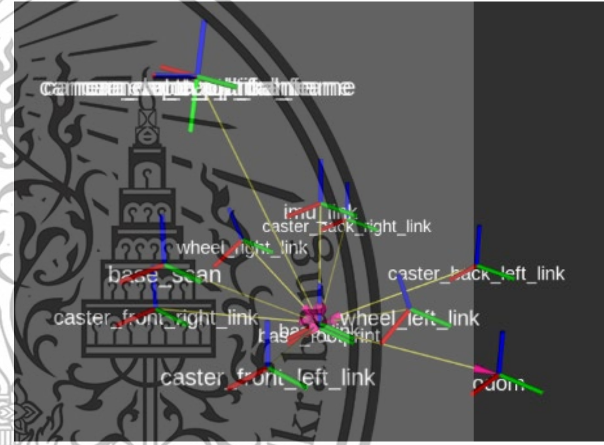


Figure II.II Seagate mobile robot 3D model Tf

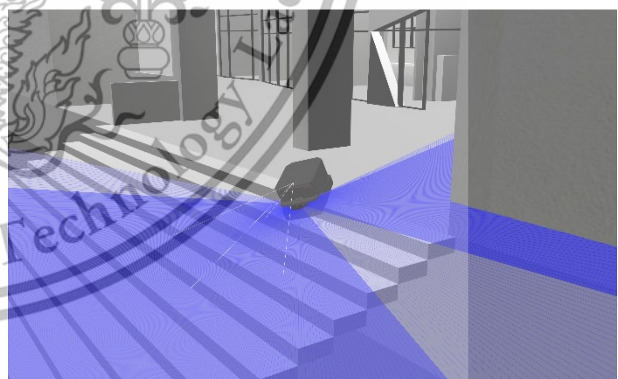


Figure II.III Seagate mobile robot 3D model

J. *ROS package migration*: The disadvantage of open-source software is the life cycle. The current ROS1 version of the Seagate Mk. II is ROS Melodic Morenia. ROS Melodic would reach EOL in April 2023 along with Ubuntu 18.04 Bionic Beaver. The migration is the adjustment of the old packages to be compatible with the newer version of ROS2 while maintaining the same functionality. Migration also ensures the further development of the robot utilizing the new features of ROS2.

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content, and cite the document when use.

K. *ROS Package Components*: A package is an organizational unit for a ROS2 code. If developers want to install the code or share it with others, then the code needs to be organized in a package. Packages enable any ROS2 code to have easier installation.

1. CMakeLists.txt: The file CMakeLists.txt is the input to the colcon build system for building software packages. Any colcon-compliant package contains one or more CMakeLists.txt file that describes how to build the code and where to install it.
2. package.xml: The package manifest is an XML file called package.xml that must be included with any colcon-compliant package's root folder. Package.xml defines the properties of a package such as the package name, version numbers, authors, maintainers, and dependencies on other catkin packages.
3. Source code: The main code no matter the format (.cpp/ .py) is required to be included in the "src" directory. It is the source code for the package.
4. Message/Service: Message and Service files (.msg/.srv) is called an Interface in ROS2.
5. Launch file: ROS 2 Launch files allow the developers to start up and configure several executables containing ROS 2 nodes simultaneously. In ROS2, the launch file can be written in .xml, .py, or. yaml format.

III. METHODOLOGY

A. *Create 3D map*: The 3D map of all 3 places mentioned earlier is generated via using Blender and Fusion360 program by the steps:

1. **Design the Map Layout**: Determine the layout and structure of the map. In this case we based on the blueprint and measurement of the real environment. Consider the environment, obstacles, and any specific features such as robot arms, television, and banners.
2. **Create the Terrain**: Start by creating the ground or terrain of the map. every Blendr project starts with creating a square mesh and modifying it to be a ground.
3. **Add Objects and Structures**: Add objects or any other elements into the map.
4. **Define Collision Boundaries**: Set up collision boundaries for objects that should interact with the robot. This will allow the robot to detect and avoid obstacles. For those 3 maps we decided that the collision boundaries are mainly walls and windows.
5. **Assign Materials and Textures**: Apply materials and textures to the objects to give them realistic appearances. The material properties can be configured by Blender's material editor to create and assign materials, and UV unwrap the objects to apply textures.
6. **Set up Lighting**: Add light sources to the map to illuminate the scene. Experiment with several types of lighting, such as sunlight or artificial lights, to achieve the desired atmosphere.
7. **Fine-tune the Environment**: Adjust the overall look and feel of the map by fine-tuning the materials, lighting, and other visual elements. You can also add details like props, decals, or particle effects to enhance realism.
8. **Export the Map**: Once satisfied with the map, the last step is to export it in a format suitable for the robot simulation. Blender supports various file formats, such as OBJ, STL, or COLLADA, which can be imported into simulation software (Gazebo).

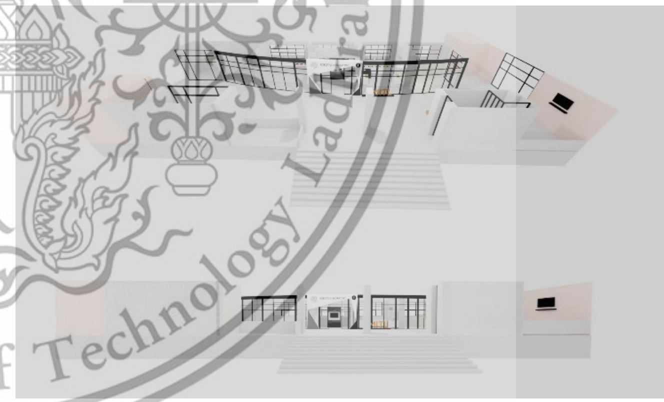


Figure III.I 3D HM Robotics lab

B. *Seagate robot for simulation*: Creating a simulation environment for the Seagate mobile robot in ROS2 Gazebo involves several essential steps. The goal is to replicate the robot's behavior and sensor outputs in a simulated setting that closely resembles the real world. Simulation enables testing and validation of the robot's algorithms, control systems, and perception capabilities before deploying in actual environments.

C. *Seagate robot modelling for simulation*: Creating a simulation environment for the Seagate mobile robot in ROS2 Gazebo involves several essential steps. The goal is to replicate

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

the robot's behavior and sensor outputs in a simulated setting that closely resembles the real world. Simulation enables testing and validation of the robot's algorithms, control systems, and perception capabilities before deploying in actual environments. The general process of creating a Seagate mobile robot simulation in ROS2 Gazebo is as follows:

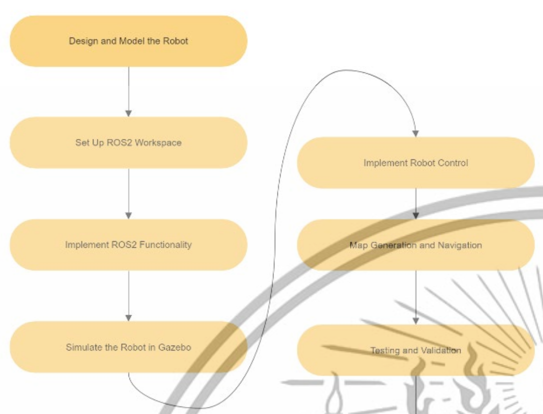


Figure III.II simulation workflow

D. **Implement ROS2 Functionality:** Configure the package dependencies to include ROS2 Gazebo and any additional packages required for the robot's specific functionality.

a. **Navigation Stack Packages**: **ros-foxy-navigation2**: The ROS2 navigation stack, which provides mapping, path planning, and localization capabilities.

b. **Sensor Simulation Packages: ros-foxy-gazebo-plugins**: Includes Gazebo plugins for simulating sensors like LIDAR and cameras.

c. **Controller Packages**: **ros-foxy-joint-state-controller**: Manages the joint states (e.g., positions, velocities) of the robot.

E. *Migration***:** The migration process of ROS package from ROS1 to ROS2 can be broken down into a series of small steps. Each step will be explained in this section.

a. **CMakeLists.txt:** The first step is reviewing the dependencies and finding their direct equivalent in CMakeLists.txt and package.xml, because these are the index of the package, it specifies which dependencies used in building and compiling the package.

i. **Cmake Version Requirements**

1. ROS1: CMake 2.8.
2. ROS2: CMake 3.5.

ii. **Find Package**

1. ROS1: “find_package(catkin REQUIRED COMPONENTS <package1> <package2> ...)”
2. ROS2: “find_package(<package1> REQUIRED)”

iii. **Include Directories**

1. ROS1: ${catkin_INCLUDE_DIRS}
2. ROS2: target_include_directories()

iv. **Link Libraries**

1. ROS1: ${catkin_LIBRARIES}
2. ROS2: target_include_libraries()

v. **Package**

1. ROS1: catkin_package()
2. ROS2: ament_package()

b. **package.xml**

i. **Format**

1. ROS1: 1, 2
2. ROS2: 2, 3

ii. **Run/execute dependencies:**

1. ROS1: run_depend
2. ROS2: exec_depend

c. **source code**

i. **Header**

1. ROS1:msg_pkg/header.h
2. ROS2: msg_pkg/msg/header.hpp

ii. **Variable:**

1. ROS1:

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

geometry_msgs::Twist var;

2. ROS2: geometry_msgs::msg::Twist var;

iii. Function Call

1. ROS1: ros::Time::now()
2. ROS2: rclcpp::Clock().now()

iv. Parameter

1. ROS1: node.param("param_name", var_param_name)
2. ROS2:This->get_parameter("param_name")

v. Publisher

1. ROS1:ros::Publisher publisher_var;
2. ROS2:Rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr var;

vi. Subscriber

1. ROS1: ros::Subscriber subscriber_var;
2. ROS2:rclcpp::Subscription<geometry_msgs::msg::Twist>::SharedPtr subscriber_var;

vii. Spin Node

1. ROS1: ros::init();
2. ROS2: rclcpp::spin();

d. launch file

i. ROS1: .xml

ii. ROS2: .xml, .yaml, .py

IV. Experimental Result

A. Simulation: The Seagate mobile robot, deployed for educational purposes and testing in simulation, has shown promising results. The robot has successfully demonstrated its capabilities in various tasks and scenarios, showcasing its potential as a versatile educational platform and a reliable testing tool.

B. navigation and path planning: The Seagate mobile robot has been able to autonomously navigate through complex environments, avoiding obstacles and following planned paths with a high degree of accuracy. The robot's robustness in handling different map configurations and dynamic obstacles has been demonstrated through extensive testing.

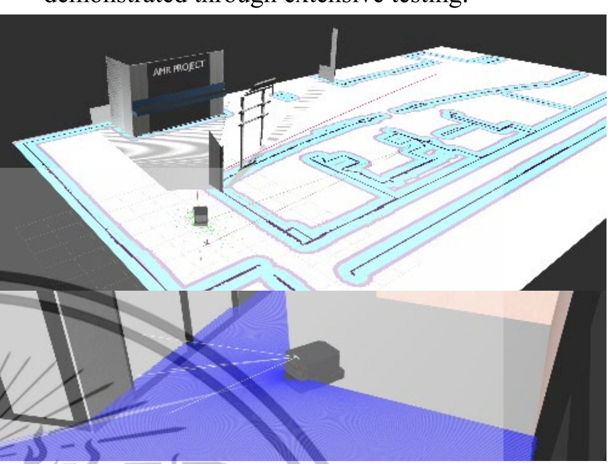


Figure IV.1 Seagate mobile robot simulation running navigation

C. migration All ROS1 packages have been successfully migrated to ROS2 Foxy and can be used as a base package for further development in the next ROS2 distro releases.

a. Topic Monitoring: The "rqt_graph" is a tool providing GUI for topic monitoring.

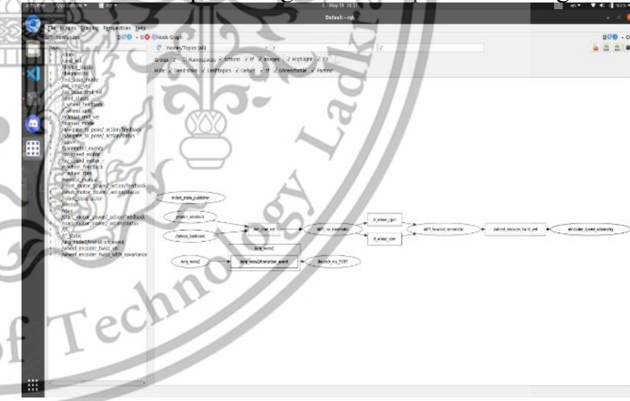


Figure IV.2 Topiv Monituring via rqt_graph

V. Conclusion

The deployment of the Seagate mobile robot in simulation has allowed for the creation and evaluation of detailed 3D maps. These maps not only serve as a valuable tool for simulating and testing robot behavior but also have the potential for broader applications. The 3D map models can be further developed and adapted for various scenarios, such as augmented reality (AR) and virtual reality (VR) tours.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

The map's compatibility with AR/VR technologies opens possibilities for creating immersive experiences and virtual tours in diverse environments, including historical sites, museums, or architectural walkthroughs. Users can explore and interact with the virtual representation of the real-world environment, leveraging the detailed 3D map model created during the Seagate mobile robot's simulation.

The migration from ROS1 to ROS2 is a crucial step in advancing the Robot Operating System. This thesis has explored the migration process in detail, covering the necessary modifications to CMakeLists.txt, package.xml, and source code.

By following the steps outlined in this research, developers can seamlessly modify their ROS1 packages to ensure compatibility with ROS2. This smooth transition results in a ROS2 codebase that can be easily migrated to future distro versions, extending the package's maintenance lifespan.

Overall, the experimental results showcase the Seagate mobile robot's capabilities as an educational and testing platform in simulation and real-world. The robot's navigation, sensing, teleoperation, and map development functionalities provide a solid foundation for educational purposes and extend to broader applications in the field of virtual tours and immersive experiences.

ACKNOWLEDGMENT

We would like to take this opportunity to express our utmost gratitude to our beloved advisor, Dr. Poom Konghuayrob, for their invaluable guidance, support, and mentorship throughout the duration of our project. Under their esteemed supervision, we were able to apply the knowledge and skills acquired during our 4-year study at KMITL (King Mongkut's Institute of Technology Ladkrabang) in a practical and meaningful manner.

Dr. Poom Konghuayrob, your unwavering commitment to our success has been a constant source of inspiration. Your expertise in the field and dedication to teaching have not only enhanced our technical understanding but also broadened our horizons. Your open-door policy and willingness to provide guidance and constructive feedback have empowered us to overcome challenges and achieve our goals. Dr. Poom's guidance was instrumental in shaping the trajectory of our project, the SEAGATE Robot Mk. II. Your keen insights and ability to think critically enabled us to navigate complex problems and develop innovative solutions. Your encouragement to explore new avenues and push the boundaries of our capabilities has fostered a spirit of curiosity and intellectual growth within us.

In addition, we would like to extend our heartfelt appreciation to our co-advisor, Mr. Sarucha Yanyong, for his effort in supporting us and providing applicable feedback throughout the course of our project. Mr. Sarucha Yanyong's dedication to our development, and willingness to invest your time and expertise have been truly commendable. Your attention to detail and commitment to excellence have helped refine our work and ensure its quality.

REFERENCES

[1] Brush, K. (2019) What is a mobile robot? definition from whatis.com., IoT Agenda. Available at: https://www.techtarget.com/iotagenda/definition/mobile-robot-mobile-robotics

[2] Why Ros? ROS. Available at: https://www.ros.org/blog/why-ros/

[3] ROS Wiki ros.org. Available at: http://wiki.ros.org/melodic

[4] Ros2 -how is it better than ROS1 (2022) Medium. Available at: https://medium.com/@oelmofty/ros2-how-is-it-better-than-ros1-881632e1979a

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content, and cite the document when use.