Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

MasterArbeit

# Study and implementation of LiDAR-based SLAM algorithm and map-based autonomous navigation for a telepresence robot to be used as a chaperon for smart laboratory requirements

Eram Arfa.

| | |
|---|---|
| **Course of Study:** | INFOTECH |
| **Examiner:** | Prof. Dr. Marco Aiello |
| **Supervisor:** | Nasiru Aboki, M.Sc. |
| **Commenced:** | October 1, 2021 |
| **Completed:** | March 31, 2022 |

## Abstract

The field of smart autonomous systems has shown rapid growth in the past years, which has led to the development of robots for human assistance. Here, the main focus of the autonomous telepresence robot is to assist as a chaperon in a smart laboratory setup. The goal is to add functionalities to a telepresence robot to help the people in the lab with various everyday tasks like controlling the intelligent lights, doors and windows, and thermostat adjustments. The robot should follow the standard protocols inspired by humans, specifically perception, thinking, and reaction. Simultaneous Localization and Mapping (SLAM) is a significant problem, where the objective is to make the robot cognizant of its position in the surroundings while building the map of the environment. There are many answers to the SLAM problem developed in the past two decades, and in this Master thesis, we study Hector SLAM and implement a solution for map-based autonomous navigation of the robot to enable it to reach a goal position. We also compare the algorithm's performance by testing how accurately the robot navigates using the map generated by the algorithm in the actual environment of the laboratory.

We have a Telepresence Robot from the Ohmni labs, modified to integrate a ydlidar and a Raspberry pi for the test bench. The robot performs navigation using the hector SLAM algorithm and metrics for localization and mapping. The robot explores the indoor environment and at the same time generates a map of it. It uses the map to localize itself in the environment and reach a goal point given to it by a custom script. The algorithm produces a relatively accurate map of the environment while measuring the robot location consistently for various speeds. The robot localizes itself accurately in two different environments and navigates to given goal coordinates. The results from the thesis are accurate and we get essential environmental perceptions (map of the surrounding, ability to localize the robot, optimal driving velocity of the robot )that can make future development work on the Ohmni robot easier. The thesis opens up the possibility of implementing motion and path planning use cases on the robot as the navigation stack of the robot is configured correctly in this work.

# Contents

# List of Figures

# List of Tables

# Acronyms

**ADB** Android Debug Bridge. 33

**AI** Artificial Intelligence. 13

**AMCL** Adaptive Monte Carlo Localization. 4

**DARPA** Defense Advanced Research Projects Agency. 17

**GPS** Global Positioning System. 18

**HRC** Human-Robot Collaboration. 57

**IAAS** Institute for Architecture of Application Systems. 11

**IMU** Inertial Measurement Unit. 19

**IP** Internet Protocol. 45

**LiDAR** Light Detection and Ranging. 16

**MLE** Maximum Likelihood Estimate. 30

**OGM** Occupancy Grid Mapping. 42

**OS** Operating System. 15

**PC** Personal Computer. 35

**PGM** Portable Gray Map. 46

**QA** Quality Assurance. 26

**RGB** Red Green Blue. 20

**ROS** Robot Operating System. 14

**ROSRPC** ROS Services and Parameters. 23

**RRT** Rapidly Exploring Random Tree. 19

**SDRAM** Synchronous Dynamic Random Access Memory. 36

**SFM** Structure from Motion. 20

**SLAM** Simultaneous Localization and Mapping. 3

**SSH** Secure Socket Shell. 22

**TF** Transform Frame. 24

**UAVs** Unmanned Aerial Vehicles. 17

| | |
|---|---|
| **URDF** | Unified Robotic Description Format. 38 |
| **URI** | Uniform Resource Identifier. 23 |
| **USB** | Universal Serial Bus. 35 |
| **VNC** | Virtual Network Computing. 36 |
| **XML** | Extensible Markup Language. 38 |
| **YAML** | Yet Another Markup Language. 23 |

# Acknowledgements

Firstly, I would like to acknowledge the opportunity that professor and mentor, Prof Dr. Marco Aiello, provided me. His constant faith and guidance towards the project kept me from not giving up. I started my work on the Telepresence robot with requirement analysis and use cases creation for my study project. The time spent over two semesters was full of learning. Dr. Aiello has always emphasized that work gets better when we collaborate. This is something I will try to inculcate in my life. I want to thank Mr. Nasiru Aboki for constantly guiding me to the right path when I was stuck. Working at the Institute for Architecture of Application Systems (IAAS) lab was a journey I will always cherish. I would also like to extend my gratitude to Vinayak Patil and Adityakrishna Okade, my batchmates working on different use cases for the robot for their respective Research Projects. It was constructive with the two of them being around and helping in every way possible. We have spent hours together during the initial days of the project to understand the robot's working.

Many people outside of my educational group have supported me along the way and had confidence in me. Anuraag Pandey, Shahbaz Akhtar, Shazia Afreen, and Ruschil Ray are a select group of people who are always willing to lend a helping hand. Honestly, I cannot ask for better friends. And finally, my mother and my sisters for encouraging me and supporting me in my dreams to travel all the way from India for a master's degree. A sincere thanks to everyone for being a part of this wonderful journey.

# 1 Introduction

Service robots used in assistance technology plays a crucial role in our everyday lives. It not only compensates for the inevitable delay which occurs due to involvement of human beings but also improves living in targeted performance areas. A telepresence robot helps “place” people at a remote location, thus providing a virtual presence. The robot available at the IAAS laboratory is a flagship product from Ohmni labs which promises to transform how people connect with each other. A telepresence robot is a wheeled mobile device which has wireless internet connectivity. It can be controlled by a computer, tablet or a smart phone. This enables the individuals who are engaged with the robot to hear and watch the operator of the robot on the display screen and vice versa. However these robots are much more than virtual presence robots or remote controlled devices.

The collaboration between and humans and robots has been a topic of both fiction and academic conjecture. The three laws of robotics dictates the following:[soulea]

- A robot may not injure a human being or, through inaction, allow a human being to come to harm.
- A robot must obey the orders given it by human beings except where such orders would conflict with the First Law.
- A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

The above three laws ensure the safety of humans while they interact with the robots. All sorts of developments performed in the field of robotics must keep the three laws into consideration. However, the field of robot ethics and machine ethics are way more complex than just these laws. Initially robots required some human involvement to function but studies have expanded this to an extent that fully autonomous systems are now far more common. Autonomous systems include from simultaneous localization and mapping systems which provide robot movement to natural language processing and natural-language generation systems which allow for natural, human-like interaction.

The goal of the thesis is to introduce certain functionalities in the robot system so that it can perform various complex tasks on its own. The innate functionality of the robot to be able to provide virtual presence for people, will not be used in this case. Instead its onus is shifted to provide assistance to the people working in the lab. For this purpose, complex interactions of the robot with other components of the lab need to be implemented, we need to establish communication with its internal operating system (the ability to ping the operating system of the robot), send commands to it, enable movements of its various parts and finally perform SLAM and provide a certain degree of autonomy. The question that arises is why was it deemed necessary to carry through the aforementioned features on the robot. The reason being, to reduce human effort and improve work efficiency or throughput by enabling human-robot interaction in the laboratory. If the employees working at the lab are spared from doing mundane tasks like closing the door, switching on/off the lights and assisting visitors coming into the lab, they would be far more productive in their work.

One of the major challenges is building the navigation system for the robot as its existing operating system is completely unknown. The first task is to understand if the internal navigation configuration set up exists and if it does not then customise it to include our functionality.

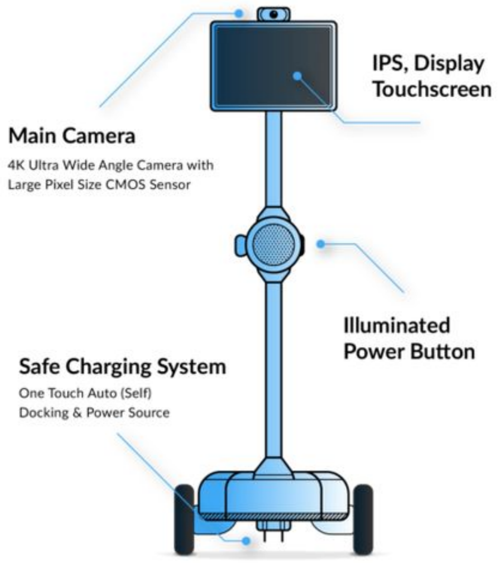

IPS, Display Touchscreen
Main Camera
4K Ultra Wide Angle Camera with Large Pixel Size CMOS Sensor
Illuminated Power Button
Safe Charging System
One Touch Auto (Self) Docking & Power Source

**Figure 1.1:** Ohmni Telepresence Robot - the Ohmni® Robot, is an award-winning telepresence robot that transforms how people connect, from their homes, businesses, classrooms, to hospitals.[souleb]

## 1.1 Motivation

We have all thought at some point in our lives of a dystopian future of work where robots would arise and take up all our jobs. But what if we live in a world where the robots take up our mundane jobs that no one likes to do. Most of us would jump at the chance to hand over our daily chores to the robot helper, and thus freeing up some time from our schedules. Now more people are inviting robots to their workspace and even homes to assist them. The ongoing pandemic has shown that human-to-human interaction must be limited in all sectors of work. Robots are a promising way to overcome this problem in assisted living environments. The Artificial Intelligence (AI) and machine learning will give a way for intelligent robots with cognitive abilities while enabling them to be more aware of their surroundings. The concept of assisted living has been widely benefited by the healthcare sectors and especially for people who are immunocompromised and must be protected. It has a high potential in the pharmaceutical industry as the robots can perform many chores with high speed and precision. Smart laboratory is a growing field of robotics. Robots handle test tubes, sort them and even perform experiments. Rush LaSelle, (Vice President and General Manager of

Adept Technology Inc) says robots provide researchers more time to perform high- level duties. “High incidence of error is a byproduct of repetitive tasks being performed manually. Robotics alleviates that potential for error.”

Considering all the factors, the IAAS department purchased the Ohmni Telepresence robot to extend its features to support the scientists in the lab with their daily tasks. The vision is to use the robot as a chaperon that connects to the smart devices in the lab. Thus becoming a hub to operate the lights, surveillance system, thermostat, and connected multimedia devices using IoT, offer emotional support, learn new things on its own, and help save energy.

## 1.2 Problem Statement

As mentioned in the previous section, mobile robot localisation and navigation have played a significant role in increasing the scalability and flexibility of the work space, but it is known that it depends on an environment with limited dynamics and modifications. A major aim of the thesis is the development of a navigation system for the Ohmni Telepresence Robot using recent contributions for solving the famous SLAM problem, as a result of which the robot is able to gain a certain degree of autonomy. Here the main focus lies on localisation and mapping of the indoor lab environment using an obstacle detection sensor and a computer. The thesis’ work bases on the following assumptions:

- The sensor used in collecting information of the environment delivers sufficient information to generate and update a map which fulfils the requirements of localisation and mapping.

Based on the above assumption the research questions that this work tackles is the following :

- **Research Question 1:** How do we establish a connection with the internal operating system of the robot and then control it with an external computer by sending/receiving commands ?
- **Research Question 2:** Is it possible to integrate the available sensors into the robot’s system and to use the information collected to generate up-to-date maps of the laboratory, which can allow the robot to operate autonomously within the said environment?
- **Research Question 3:** Considering localization and mapping are closely related, what will be the integrated approach needed that can both provide robust and accurate localization for a robot and create a global map utilizing all available sensor information?

## 1.3 Research Methodology

The research work intends to have an inference on using a non-odometric approach of Hector SLAM as it is one of the most used SLAM algorithms in Robot Operating System (ROS) framework. A part of the research focuses on the evaluation of the results against different performance parameters. It is further extended by using the latter’s results to achieve autonomy of the robot and finally analysing the performance. The plan employed to resolve the aforementioned research questions is as follows :

- As it has been already established that SLAM is a solved problem therefore an extensive literature is available. For the given test bench (the telepresence robot), SLAM or map-based navigation has not been implemented before hence it was of utmost importance that previous work of SLAM implementation on similar differential drive robots be referred, to pave the way for the author to carry out the work. It adds value to the legitimacy of the work by helping adopt a more appropriate methodology for the research. It elucidates the strengths and weaknesses of existing research in the same field.
- Familiarization with ROS Framework and Docker were the areas that required deep investigation and thus aided as the starting point of the formal work. The Ohmni Developer Edition is built with a powerful Docker virtualization layer that can be utilized by the ROS developers to run the ROS framework on top of Ohmni OS. There is a default ohmnilabs/ohmnidev image and another docker image, tb_control for simplified development.
- System Design and defining the hardware and software components required is a crucial process for any development work. A system is an organized relationship between different component groups that work together to achieve a common goal. A meticulous breakdown of the system into its components helps in the identification of the various dependencies between them and failures which might occur if some of the parts stop working. It clarifies the goal, improves the quality of work, and occurrences of errors can be reduced considerably.
- Once the system architecture was ready, the next step was to start setting up the required hardware components. The main hardware component is the telepresence robot itself, any other hardware component to be integrated with the former had to be compatible with it. Before coming to any conclusion, it was important to assess the needs of the tasks that were planned to be computerized for example, the operating system to support the development, the middleware framework needed to run the algorithms, and the virtualization environment to perform the simulations. All of these demands some of sort of specifications like processor speed, model, manufacturer, memory size and so on.
- After successfully deciding on the hardware components to be used, the next step was the software setup and decisions on the algorithms and already available software packages to be utilized as a part of the implementation work. Also, it's crucial to decide at the very beginning the high-level programming language in which the development would be done. Ohmni robot runs on OhmniLabs' custom version of Android-x86 called the Ohmni Operating System (OS). It is a hybrid between Android and Ubuntu that combines features of Android such as the familiar interface and boot process of a full Ubuntu system. Ohmni OS runs a Linux kernel. It was essential to set up the software components in alignment with the ones pre-installed.
- Implementation begins once the hardware components are wired together and software setup is completed. To confirm if everything was running, it was necessary to run a sample piece of code to validate the connections. Once this step is finished, the actual development starts, and the required functionalities are broken down into smaller tasks for example creating a map of the environment, localizing the robot on the map, setting up the robot navigation stack and so on.
- To check the final code, verification and validation techniques are used. A number of test cases are run to optimize the configuration files against the performance metrics and assess the results.

## 1.4 Organization

The rest of the thesis is structured as follows.

- Chapter 2 provides the reader with required background into research related to autonomous robots, and of related research work that is currently being pursued. It introduces SLAM and describes the range of the various Light Detection and Ranging (LiDAR) based SLAM algorithms tested and compared.
- Chapter 3 puts light on the state of the art and the necessary literature survey done during the course of the thesis. It also gives an overview of the test bench, i.e., the telepresence robot and Robotic Operating System (ROS) framework.
- Chapter 4 discusses the system components, the overall hardware interconnections and set up of the navigation stack of the robot. It gives a detailed description of the configuration files and the robot model definition.
- Chapter 5 provides the experimental setup and results for the tested algorithm, details of the map-based navigation and the evaluation techniques used to confirm the results.
- Finally, the thesis in concluded in Chapter 6 with a summary of results.

# 2 Background

## 2.1 Background

### 2.1.1 Autonomous Systems

The expression “Autonomous”, reminds us of industrial automation, Unmanned Aerial Vehicles (UAVs) , self-driving cars and so on. These are modern technologies with ancient roots. The concept of autonomous systems dates back centuries ago with the term “airscrew” coined by Leonardo Da Vinci [FOL76], also known as the Da Vinci’s Helicopter. Alan Turing developed the Bombe during the second world war. It helped decipher the German’s secret communications - Enigma. He also invented the Turing Machine called the “a-machine” or “automatic machine” [HT12]. The Turing Machine is the conceptual forerunner of the modern computer and helps explain the idea of automation. Every modern technology uses the theory of automation in some form or the other. A simple “for-loop” statement in a computer programming language is an example of automation. Current applications of autonomous systems deal with solving everyday problems, like efficient conveyance without human interference, e.g., autonomous vehicles[Urm+08], cleaning robots like Roomba by iRobot [FD06], warehouse automation in terms of precise manufacturing, warehouse organization [KUTY00] , and delivery using robots [ZZZ14].

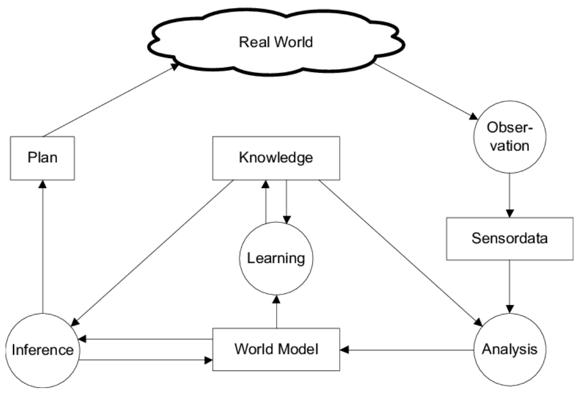


**Figure 2.1:** General information flow in autonomous systems

1. **Self Driving Cars**: Self-driving cars (also known as autonomous vehicles or driverless vehicles) are a popular research topic in the automotive industry and academia right now. Hundreds of thousands of hours of research have gone into bringing the best technology to market. Companies like Tesla, Uber, Aptiv, NuTonomy, and others have recently come close to commercializing the technology. The research began even before the Defense Advanced Research Projects Agency (DARPA) Grand Challenge in 2004, when all of the competing

cars failed during the first miles, but they realized a platform to showcase their autonomous car research. In 2005, the challenge was repeated, with Stanford University's car Stanley taking first place.[TMD+06]

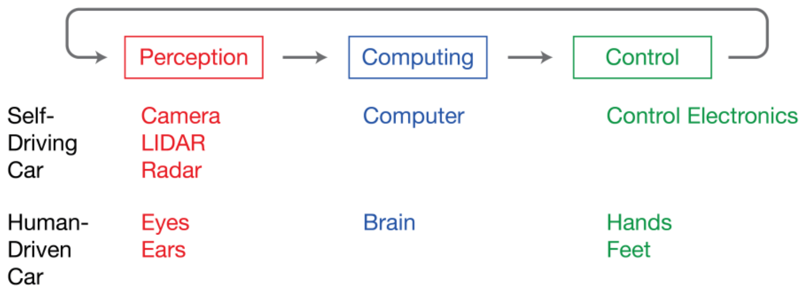


**Figure 2.2:** A self-driving car, like that of a human driver, goes through the process in which it perceives the situation, considers it, and then acts based on its assessment. High-tech devices perform each of these tasks.

2. **Warehouse and Delivery Robots:** Warehouse and delivery robots are one of the engineers' solutions to the supply-chain management and logistics problem. The need for robotics in logistics is stated in [ROB16]. These robots are able to navigate the warehouse either by learning the map or by using a pre-programmed map. Because there is no Global Positioning System (GPS) signal available indoors, unlike self-driving cars, GPS is not an option. The robots estimate their position using beacons or visual/colored landmarks, that are used to solve the localization problem. Inside the warehouse, the robots can pick, drop, and sort packages.Because warehouse maps are typically static, with defined locations for shelves and stacks for picking and dropping, path planning is as simple as implementing one algorithm and dynamically updating the path to avoid any moving obstacles in the way.

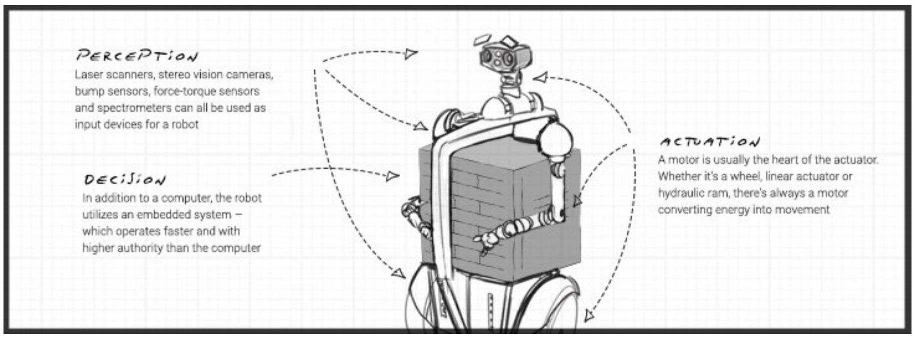

PERCEPTION
Laser scanners, stereo vision cameras, bump sensors, force-torque sensors and spectrometers can all be used as input devices for a robot
DECISION
In addition to a computer, the robot utilizes an embedded system – which operates faster and with higher authority than the computer
ACTUATION
A motor is usually the heart of the actuator. Whether it's a wheel, linear actuator or hydraulic ram, there's always a motor converting energy into movement

**Figure 2.3:** A fully autonomous robot is something that can perceive its environment, decide based on what it perceives, and/or has already been programmed to recognize conditions, and then act on those perceptions by moving or manipulating within that environment.

### 2.1.2 Common Pipelines

Every self-driving robot produced, adheres to a similar control pipeline. Because the control system is a closed-loop with feedback, the autonomous robot requires sensors. Figure 2.1 depicts the common control pipeline that is at the heart of every autonomous robot. There are two types of sensors that are often used:

- **Exteroceptive Sensors:** Measure the state of the environment (mapping, temperature, etc.). Example: LiDAR, Camera, Ultrasonic Sensor, Radar.
- **Proprioceptive Sensors:** Measure the state of the robot itself (wheel position or speed, battery charge, etc.) Example: Encoders, Inertial Measurement Unit (IMU).

Simply put, perception is what the sensors see (perceive) from their surroundings. Sensors can detect a moving person (dynamic object) or a picture on the wall (stationary object or Landmark).These observations are utilized to create an environment map. Static objects (or landmarks) are part of the global map, but dynamic objects are employed by the local planner to create a path around them by estimating their motion.

Localization is another component of the pipeline. The perception algorithm's landmarks are utilized to learn a map. These markers are also utilized to calculate the robot's pose (position and orientation) in the created map. Unlike self-driving vehicles, where the GPS receiver is a primary source of location data, indoor robots used for warehouses navigation and vacuum cleaning robots cannot depend on GPS for navigation and localization. Several techniques for non-GPS-based localization have been proposed. Such algorithms use sensors like LiDAR and cameras to understand the environment and predict the robot's pose. [LT10] proposed a localization method based on an offline grid map of probability distributions over environment reflectance to the LiDAR's laser rays. Some solutions rely on LiDAR data to construct a map and camera data to estimate the location of the self-driving car in relation to the generated map.

Once the map has been constructed and the robot has been localized, the next stage is decision making, which entails designing a path to the target while also taking into account the holonomic constraints of the robot. Path planning algorithms such as Dijkstra's, A*, Rapidly Exploring Random Tree (RRT), and RRT* are employed. The SLAM problem is built by combining the localization and mapping problems. The goal is to consolidate the localization and mapping pipelines into a single SLAM algorithm. The most discussed problem in the realm of robotics is SLAM, or Simultaneous Localization and Mapping. DurrantWhyte and John J. Leanord first proposed the topic in 1991 [LDW91]. The problem specifies that if a moving robot is placed in an unknown surrounding, how probable is it to construct a coherent map of the observed environment while measuring its present location and navigating through it?

Various solutions to SLAM have been proposed over the last two decades, and it is currently considered solved. Some solutions necessitate the use of many sensors, whereas others necessitate the use of only one sensor. Algorithms are developed to interpret the incoming data from these sensors and create a map by monitoring the surroundings, identifying landmarks, and approximating the location of the moving platform in relation to the landmarks. Any ambiguity in mapping and/or localization is frequently caused by noisy sensor data. Uncertainty leads to an erroneous perception of one's surroundings. Localization and Mapping can be discussed separately by the following equations :

$$\text{Localization : p(x|z ,u)}$$

$$\text{Mapping : p(m|z ,u)}$$

Where m is the environment map, x is the robot's current pose in the world frame, z is the observation, which is information from sensor measurements, and u is the robot's control command or odometry. As shown below, these elements combine to generate the SLAM problem. [GKSB10]
Given,

$$\text{Sensor Readings: } z_{0:T} = z_0, z_1, z_2, ..., z_T \text{ and}$$

$$\text{Odometry: } u_{1:T} = u_1, u_2, u_3, ..., u_T$$

Find,

$$\text{Posterior: p}(x_{0:T}, \text{m}|z_{0:T}, u_{1:T})$$

SLAM is solved over time (0:T), and the result is given as a conditional probability of the path (x0:T: pose over time) and map (m), given sensor readings (z0:T), and control and odometry (u1:T) over time. This probability is usually Gaussian, which aids in the better estimation of unknown situations over time. Many LiDAR-based SLAM algorithms employ Monte Carlo-based Kalman Filters or Particle Filters [Gri05] and [Gri07] (also known as Sequential Monte Carlo), whereas Visual (camera-based) SLAM employs Structure from Motion (SFM) techniques to estimate camera pose and motion in the environment by processing multiple images of the same environment taken from different angles.

### 2.1.3 Sensors

To try to tackle the SLAM challenge, a number of sensors have been utilized. Exteroceptive sensors can be classified based on the number of sensors or the type of data. There are short range sensors, such as ultrasonic sensors, and long range sensors, such as LiDAR and cameras. Sensors give color (Red Green Blue (RGB)) images, depth images, 2D point cloud, and 3D point cloud for perception and mapping, depending on the type of input. Proprioceptive sensors for measuring the odometry of a moving platform include encoders, an IMU, and GPS. The strength of the received signal from beacons is also used to locate a robot on a map. The beacons serve as benchmarks for estimating the robot's pose.

## 2.2 ROS - Robot Operating System

### 2.2.1 Fundamentals of ROS

Most of the information presented in this section is referred from [QGS15]. ROS is an open-source framework to send out commands to the robot and make them perform certain tasks. It provides a common software platform for people who want to build robots and program them, share codes and ideas and do not spend years building a software architecture to make the robot move.

ROS consists of the following parts :

- A number of software drivers that are capable of reading sensor data and send commands to the actuators/motors of the robot.
- An extensive multitude of algorithms to support various functionalities like building maps, navigation, localization, manipulate objects and lot of other stuffs.
- A large set of tools to enable visualization of the robot state and validate the algorithms running in the system.
- Finally, ROS is an open-source so there is a large community of developers and users that can provide solutions to a spectrum of problems which may arise during the work. There is enough documentation available to walk one through the various stages of development.

#### 2.2.1.1 TheROSGraph

A ROS System is made up of a collection of programs that run simulateneously and communicate with each other by passing messages. A mathematical graph is used to depict the communication between the nodes i.e. the programs and the messages. Such a graph is called ROSGraph and it is very helpful when one wishes to view all the programs herein called nodes at a glance. Any ROS system can be represented using the ROSGraph. Thus, the ROSGraph node is a software module, the graph edge is the message exchange between the nodes.

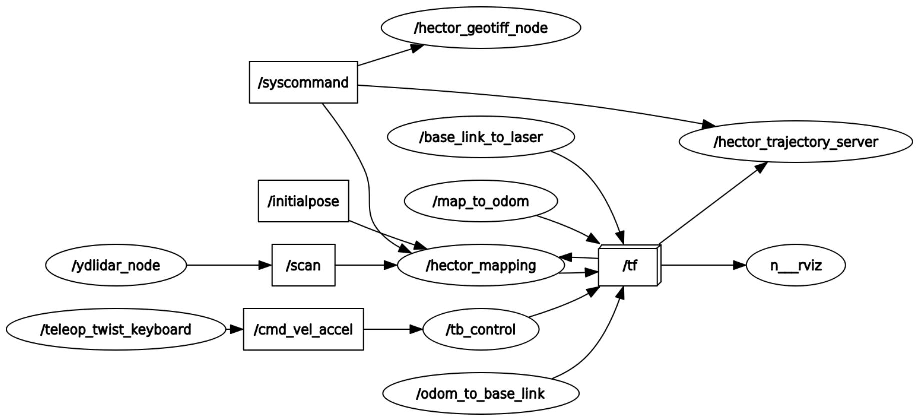
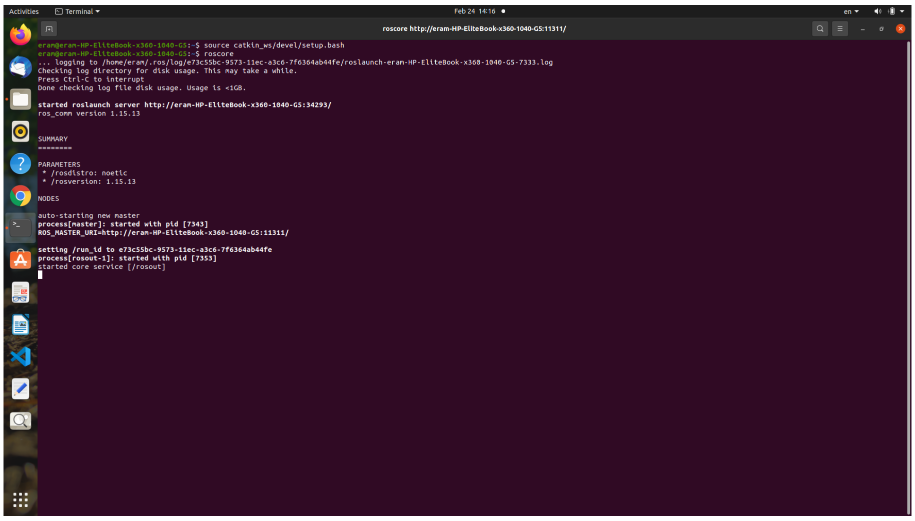


**Figure 2.4:** A ROSGraph after running the teleoperation node (teleop twist keyboard package) and Hector SLAM package on the Ohmni Robot

#### 2.2.1.2 ROS Commands-Line Tools

1. **roscore :** roscore is a collection of nodes and applications that are necessary for a ROS-based system to operate. In addition for ROS nodes to communicate, a roscore must be running. The roscore command is used to launch it.

   If one uses roslaunch, it will start roscore immediately if it detects that it is not already operating. **roscore** starts the ROS Master, a ROS parameter server, and a rosout logging node.



```
eram@eram-HP-EliteBook-x360-1040-G5:~$ source catkin_ws/devel/setup.bash
eram@eram-HP-EliteBook-x360-1040-G5:~$ roscore
... logging to /home/eram/.ros/log/e73c55bc-9573-11ec-a3c6-7f6364ab44fe/roslaunch-eram-HP-EliteBook-x360-1040-G5-7333.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://eram-HP-EliteBook-x360-1040-G5:34293/
ros_comm version 1.15.13


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.13

NODES

auto-starting new master
process[master]: started with pid [7343]
ROS_MASTER_URI=http://eram-HP-EliteBook-x360-1040-G5:11311/

setting /run_id to e73c55bc-9573-11ec-a3c6-7f6364ab44fe
process[rosout-1]: started with pid [7353]
started core service [/rosout]
```

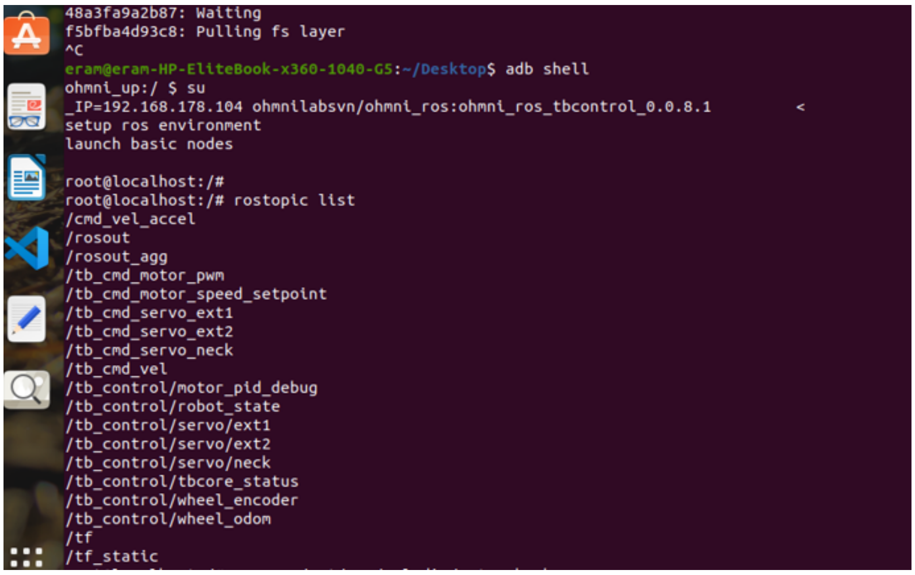
**Figure 2.5:** It is essential for roscore running so that the ROS nodes can communicate with each other

2. **rosrun :** rosrun allows you to use the package name to directly run a node within a package (without having to know the package path).[soulec]
3. **rosnode :** Displays debugging information about ROS nodes, including publications, subscriptions and connections. Commands:
   - rosnode ping : Test connectivity to node.
   - rosnode list : List active nodes.
   - rosnode info : Print information about a node.
   - rosnode machine : List nodes running on a machine.
   - rosnode kill: Kill a running node. [souled]
4. **roslaunch :** This command line tool is used to start a roscore (if required), a number of local nodes, remote nodes through Secure Socket Shell (SSH), and sets parameter server.
5. **rosservice :** A tool for listing and querying ROS services. Commands:
   - rosservice list: Print information about currently active services.
   - rosservice node: Print the name of the node that is delivering a service.

- rosservice call: Use the specified args to call the service.
- rosservice args: Returns a list of a service's arguments.
- rosservice type: Displays the service type.
- rosservice uri: Print the ROS Services and Parameters (ROSRPC) service Uniform Resource Identifier (URI).
- rosservice find: Locate services according on their type.

6. **rostopic :** The command-line utility rostopic displays information about ROS topics [soulee]. It may currently provide a list of active topics, the publishers and subscribers of a single topic, a topic's publication pace, bandwidth, and messages written to a topic. Message display can be configured to output in a plotting-friendly format.

   At the command line, rostopic, like several other ROS utilities, employs Yet Another Markup Language (YAML)-syntax to represent the contents of a message.



```
48a3fa9a2b87: Waiting
f5bfba4d93c8: Pulling fs layer
^C
eram@eram-HP-EliteBook-x360-1040-G5:~/Desktop$ adb shell
ohmni_up:/ $ su
_IP=192.168.178.104 ohmnilabsvn/ohmni_ros:ohmni_ros_tbcontrol_0.0.8.1        <
setup ros environment
launch basic nodes

root@localhost:/#
root@localhost:/# rostopic list
/cmd_vel_accel
/rosout
/rosout_agg
/tb_cmd_motor_pwm
/tb_cmd_motor_speed_setpoint
/tb_cmd_servo_ext1
/tb_cmd_servo_ext2
/tb_cmd_servo_neck
/tb_cmd_vel
/tb_control/motor_pid_debug
/tb_control/robot_state
/tb_control/servo/ext1
/tb_control/servo/ext2
/tb_control/servo/neck
/tb_control/tbcore_status
/tb_control/wheel_encoder
/tb_control/wheel_odom
/tf
/tf_static
```

**Figure 2.6:** The figure shows the available topics on the Ohmni robot after running the command rostopic list inside the Ohmni Docker

7. **rosmsg and rossrv :** rosmsg and rossrv are useful command-line utilities that offer developers with reference material as well as a robust introspection tool for learning more about data being broadcast in ROS.The command-line utility rosmsg provides information on ROS Message kinds.The rossrv command-line tool displays information about ROS services.[soulef]

### 2.2.1.3 ROS Logging Tools

1. **rosbag :** In ROS, a bag is a file format for holding ROS message data. Bags play a crucial role in ROS, and a number of tools have been built to let users to store, process, analyze, and visualize data.

ROS bags are supported via the rosbag command-line utility. It can capture a bag, republish messages from one or more bags, describe a bag's contents, verify message definitions, filter messages based on a Python expression, compress and decompress a bag, and reconstruct a bag's index.

2. **rqt_graph and ros_depth :** Tools for presenting graphs of active ROS nodes with connected topics and package dependencies.
3. **Transform Frame (TF)_echo :** A tool that prints the information about a particular transformation between a source frame and a target frame.

#### 2.2.1.4 ROS Data Visualization Tools

1. **view_frames :** A tool for visualizing the full tree of coordinate transforms.

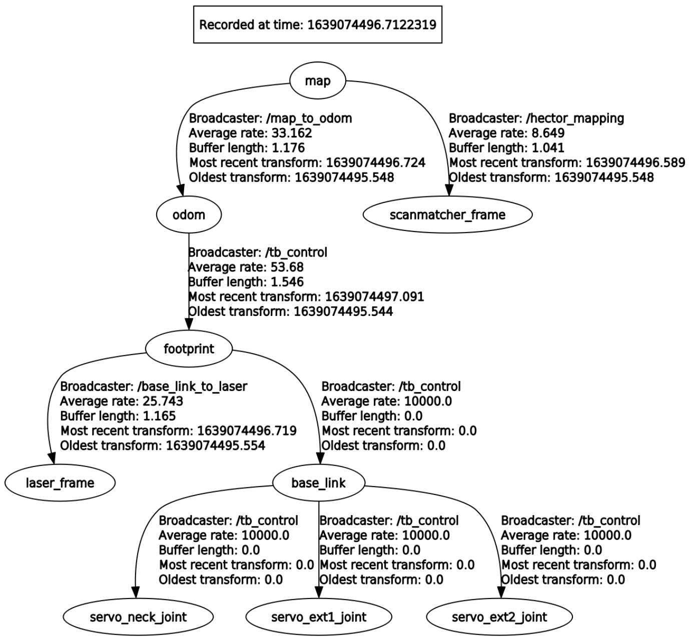


**Figure 2.7:** The figure shows the available nodes on the Ohmni robot after integrating the laser scanner and running Hector SLAM Algorithm

2. **rqt_plot :** A tool for plotting data from ROS topic fields.

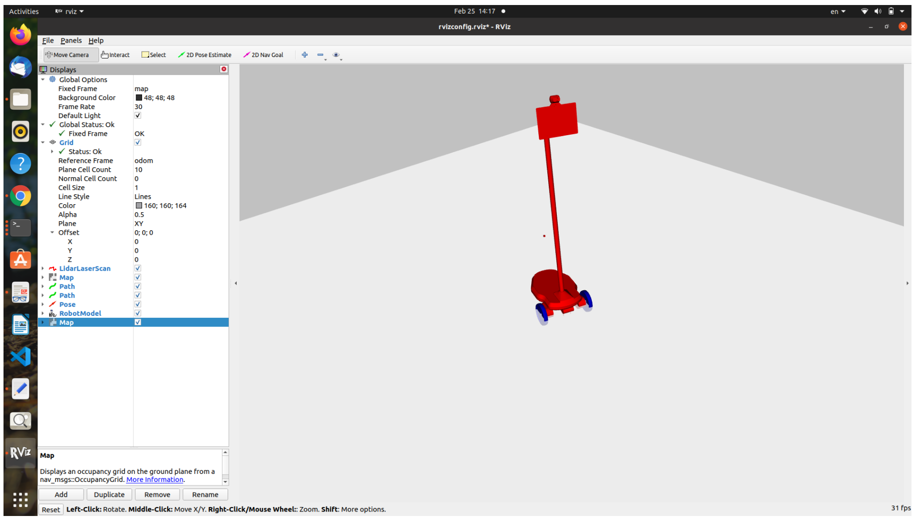
#### 2.2.1.5 catkin, Workspaces, and ROS packages

- **catkin** is a ROS build system which comprises of a set of tools that ROS uses to generate programs, libraries, scripts, and interfaces that other code can utilise. It includes CMake macros and custom Python scripts. CMake is an open-source build system and package.xml adds specific information to make things work as per the use cases.

- **Workspaces :** Before the development process begins it's important to set up the place where the code lives. A workspace is a collection of directories in which the ROS code resides. There is a possibility of having multiple workspaces but at a point in time only one can be used.
- **ROS Packages :** ROS is arranged into packages wherein each of them contain code, data and documents. Packages are available inside workspaces in the src directory. Each package must contain CMakeLists.txt file and a package.xml file.

### 2.2.2 ROS Visualization Tool

Rviz is a three-dimensional visualization tool for ROS applications. It displays the robot model, collects sensor data from robot sensors, and replays captured data. It can visualize the data from cameras, lasers, 3D and 2D devices, and also images and point clouds.
To complete the actions listed above, rviz must be open and connected to a simulation job that is running.



| | |
|---|---|
| Global Options | |
| Fixed Frame | map |
| Background Color | ■ 48; 48; 48 |
| Frame Rate | 30 |
| Default Light | ✓ |
| Global Status: Ok | |
| Fixed Frame | OK |
| Grid | ✓ |
| Status: Ok | |
| Reference Frame | odom |
| Plane Cell Count | 10 |
| Normal Cell Count | 0 |
| Cell Size | 1 |
| Line Style | Lines |
| Color | ■ 160; 160; 164 |
| Alpha | 0.5 |
| Plane | XY |
| Offset | 0; 0; 0 |
| X | 0 |
| Y | 0 |
| Z | 0 |
| LidarLaserScan | ✓ |
| Map | ✓ |
| Path | ✓ |
| Path | ✓ |
| Pose | ✓ |
| RobotModel | ✓ |
| Map | ✓ |

**Map**

Displays an occupancy grid on the ground plane from a nav_msgs::OccupancyGrid. More Information.

**Left-Click:** Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options.

**Figure 2.8:** The robot model visualized in rviz

### 2.2.3 Robotic Simulation - Gazebo

Gazebo is a 3D interactive simulator that can correctly and efficiently model robot populations in complicated indoor and outdoor situations. While similar to gaming engines, Gazebo provides far better resolution physics modeling, a range of sensors, and interfaces for both end-users and programs. The simulation model for Ohmni telepresence robot is as shown below.

## 2.3 Docker

Docker is a container management service. The important features of Docker are :

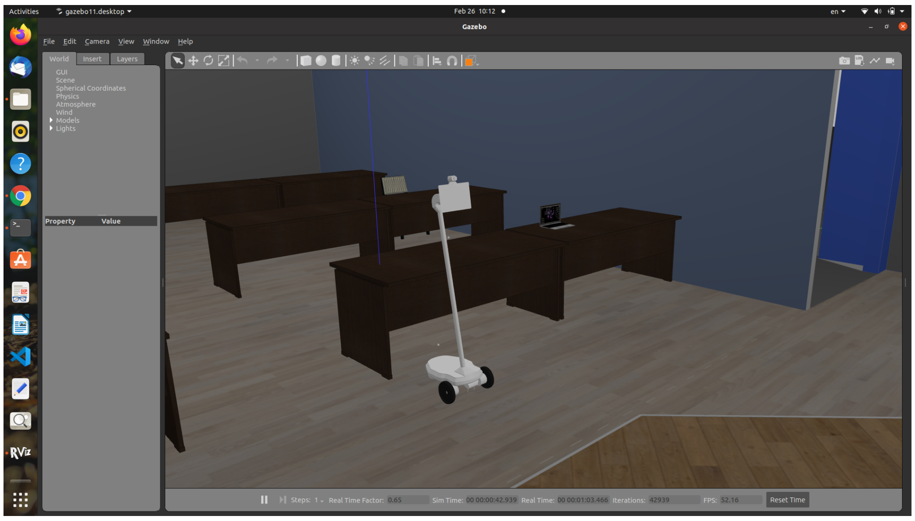


**Figure 2.9:** The robot model in Gazebo simulator designed by Ohmni Labs

- Docker has the capacity to reduce development size by offering a reduced operating system footprint via containers.
- Containers make it easier for teams from various areas, such as development, Quality Assurance (QA), and operations, to collaborate across apps.
- Docker containers can be deployed everywhere, including on virtual machines, as well as in the cloud.
- Docker containers are extremely scalable due to their small size.[poi17]

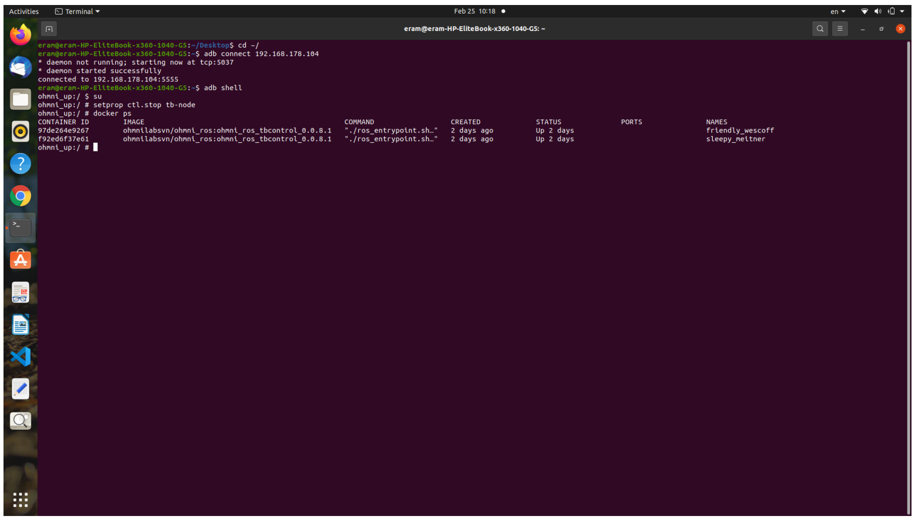
#### 2.3.0.1 Images :

Everything in Docker is based on Images. An image is made up of a file system and parameters. The Ohmni Developer Edition includes a robust Docker virtualization layer that allows any version of Ubuntu to operate inside Ohmni.

The default command to run is dockerenv, which can be executed from any adb or ssh shell. This will pull the ohmnilabs/ohmnidev image from the Docker Hub repository, which is an Ubuntu 18.04 base image with some additional utilities added.[labna]

#### 2.3.0.2 Containers :

Containers are Docker image instances that can be run with the Docker run command. Docker's primary function is to execute container.



```
eram@eram-HP-EliteBook-x360-1040-G5:~/Desktop$ cd ~/
eram@eram-HP-EliteBook-x360-1040-G5:~$ adb connect 192.168.178.104
* daemon not running; starting now at tcp:5037
* daemon started successfully
connected to 192.168.178.104:5555
eram@eram-HP-EliteBook-x360-1040-G5:~$ adb shell
ohmni_up:/ $ su
ohmni_up:/ # setprop ctl.stop tb-node
ohmni_up:/ # docker ps
CONTAINER ID        IMAGE                                               COMMAND                  CREATED             STATUS              PORTS               NAMES
97de264e9267        ohmnilabsvn/ohmni_ros:ohmni_ros_tbcontrol_0.0.8.1   "./ros_entrypoint.sh…"   2 days ago          Up 2 days                               friendly_wescoff
f92ed6f37e61        ohmnilabsvn/ohmni_ros:ohmni_ros_tbcontrol_0.0.8.1   "./ros_entrypoint.sh…"   2 days ago          Up 2 days                               sleepy_meitner
ohmni_up:/ #
```

**Figure 2.10:** List of available docker containers created on the Ohmni docker

## 2.4 LiDAR-based SLAM

Data from the surroundings (exteroceptive) as well as data from the robot (proprioceptive) are required for SLAM to function. Exteroceptive sensors such as LiDARs, cameras, and sonars are used in various ways to generate relevant data from the surroundings. Furthermore, proprioceptive data is commonly obtained via wheel encoders or sensor fusion of acceleration and orientation data from an IMU. It is critical to ensure that the SLAM algorithm receives and processes all of this data. The exteroceptive sensors are used to locate landmarks in the surroundings, which are then used to build a map by learning, and they are also used to localize within the map that has been built. After detecting and processing new landmarks, the SLAM algorithm attempts to correlate these new landmarks with any previously seen landmarks. Obstacles can act as landmarks, while landmarks can behave as obstacles. At times, the SLAM algorithm is localizing and mapping landmarks in the robot's real-time path planning.

As the robot explores and scans its surroundings, the SLAM system estimates the robot's current position inside the map using odometry data. This estimation is compared against environmental readings collected from sensors in order to permit corrections if these readings are found to be erroneous. The procedure is repeated until a comprehensive map of the environment is produced. A simple LiDAR-based SLAM algorithm is composed of five major components: landmark extraction, data association, state estimate, state update, and landmark update. The figure below gives a detailed view of the process.[RB03]

### Hector SLAM

Hector SLAM is an open-source method that uses a laser scan sensor to create a 2D grid map of the surrounding area (LiDAR). This system uses scan matching to determine the robot's location rather than the odometry of the wheel, which is a common method in other SLAM techniques. The robot's traveling trajectory is usually measured via odometry. As a result, a series of poses (positions and orientations) all pointed to an initial pose in which the odometry frame is located. It's important to

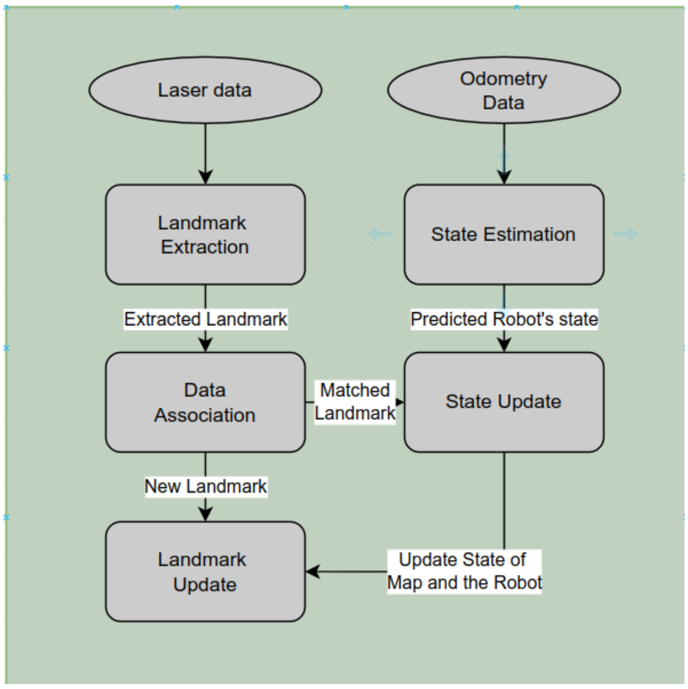


**Figure 2.11:** LiDAR-based SLAM process

note that odometry does not refer to any external or global frame like a map. When compared to the robot's real ground truth path, odometry suffers from drift due to the lack of a global reference. This drift continues to increase indefinitely throughout time. After long courses (> 1 km), more exact odometry systems may have a drift of a few cm and a few degrees, whereas less precise ones may have drifts of several meters after running a few hundred meters. Anyway, all of them will always have an unbounded drift.

Due to its rapid update rate, the LIDAR is able to complete the scan matching task to locate the robot quickly and accurately. The Hector algorithm employs the Gaussian Newton minimization approach, which is considered an update to the Newton method and has the advantage of not requiring the computation of second derivatives. By obtaining the transformation T, this method is utilized to determine the best laser scan end point in relation to the construct map.

$\zeta = (\mathrm{p}_x, p_y, \Psi)$
where,

$$\zeta* = argmin \mid ([1 - M(\mathrm{S}_i(\zeta))]^2)$$

Where M is the map and $S_i$ is the coordinates of the laser end point. After that, the error could be expressed as:

$$\mid_{i=1}^{n} ([1 - M(\mathrm{s}_i(\zeta + \Delta\zeta))]^2) \rightarrow 0$$

Then, $\Delta\zeta$ goes to zero and M is factorized using the Taylor series, therefore

$$\Delta\zeta = H^{-}1 \mid {}^{n}_{i=1} [\Delta M(\mathrm{s}_i(\zeta))(\delta \mathrm{s}_i(\zeta)/\delta\zeta)]^T [1 - M(\mathrm{s}_i(\zeta))]$$

where

$$H = [\Delta M(\mathrm{s}_i(\zeta))(\delta \mathrm{s}_i(\zeta)/\delta\zeta)]^2$$

This algorithm, on the other hand, does not have a closed- loop; it closes its loop in the real world. It has the advantage of requiring less energy to avoid changes in dynamic surroundings.

## 2.5 Localization - AMCL

### 2.5.1 Particle Filter

A large number of particles spanning the whole state-space are used to establish particle filters. As one collects more data, one predicts and updates their measurements, resulting in a multi-modal posterior distribution for the robot. This is in stark contrast to a Kalman Filter, which approximates the posterior distribution as Gaussian. The particles converge to a single value in state space after several iterations. The steps followed in a particle filter approach are as mentioned below:

- **Re-sampling:** Draw a random sample from the sample set using replacement according to the (discrete) distribution described by the important weights. This sample can be considered an example of the belief. In simple terms, draw a new set of particles with replacement from the existing (and weighted) set of particles. While doing so, give preference to particles with high weights. Particles with large weights are drawn more frequently, while particles with low weights are drawn less frequently. Maybe just once, or maybe not at all. All weights are given the same weight after re-sampling.
- **Sampling:** To sample from the distribution that describes the dynamics of the system, use past belief and control information. The density determined by the product of distribution and an instance of the prior belief is now represented by the current belief. The proposal distribution used in the next stage is based on this density.
- **Importance sampling:** The sample is weighted by the importance weight, which is the likelihood of the sample X given the measurement Z.

Each of these three procedures generates a sample selected from the posterior belief for each repetition. The relevance weights of the data are adjusted after n repetitions so that they sum to 1.

### 2.5.2 Adaptive Particle Filter for Robot Localization

The random distribution of particles throughout the state space is a significant difficulty with particle filters, which gets out of hand when the problem is high dimensional. Because of these factors, an adaptive particle filter is far superior to a simple particle filter in terms of convergent time and computational efficiency.

The core concept is to limit the inaccuracy generated by the particle filter's sample-based representation. The real posterior is assumed to be given by a discrete, piecewise constant distribution, such as a discrete density tree or a multidimensional histogram. The number of samples can be determined for such a representation so that the distance between the sample-based Maximum Likelihood Estimate (MLE) and the true posterior does not exceed a pre-specified threshold. The number of particles required is proportional to the inverse of this threshold, as determined in the end.

To utilize an adaptive particle filter for localization, we start with a map of the surroundings and either put the robot to a specific place, in which case it is manually localized, or we can start the robot from no initial estimate of its position. Now, produce new samples as the robot goes ahead, predicting the robot's position after the motion order. By re-weighting these samples and leveling the weights, sensor readings are integrated. In general, adding a few random uniformly distributed samples is beneficial since it aids the robot's recovery in circumstances where it has lost track of its position. If these random samples are not provided, the robot will continue to re-sample from an inaccurate distribution and will never recover. Because there may have dis-ambiguities inside a map due to symmetry in the map, which is what gives us a multi-modal posterior belief, it takes the filter numerous sensor readings to converge.

# 3 State of the Art

## 3.1 Literature Survey

Simultaneous localization and mapping is a broadly researched topic in robotics, and therefore a considerable amount of literature is available in the form of books and scientific papers. A primary requirement of the Master Thesis was to implement SLAM on the Ohmni robot. For that aim, the author looked at some of the previous implementations on various robots and attempted to adapt the same notions to the telepresence robot.

[LH02] explains the Intelligent Space concept. Intelligent Spaces are rooms or locations that are fitted with sensors that allow them to sense and understand what is going on around them. People or systems in the Intelligent Space can use additional functionalities provided by the space with such features. This work reflects the very purpose/goal of acquiring the robot for the laboratory. [SATS20] gives a detailed overview of a LiDAR sensor for 2D mapping construction in an unknown environment and its capability to localize itself based on landmarks detected. The LiDAR sensor is placed on a vehicle, and the robot is moved around the environment. The experiment considers both the static and dynamic obstacles entering and leaving the environment, which was a significant takeaway from this paper. The results are tested at different velocities of the robot and scanning rates of the laser sensor. However, the paper did not mention the challenges faced during the experiments conducted and only focused on the direct implementation of the SLAM algorithm. [AZ18] identifies the challenges that SLAM faces, as well as a discussion of several fundamental and contemporary SLAM algorithms. The research direction for this subject, as well as the challenges and unresolved issues, were presented in the paper. However, a novel SLAM technique had to be developed to address the difficulty discussed in the research [AZ18] . [YLC+19] presents the experimental platform and data for LiDAR-based SLAM in this work. The history of SLAM technology development and related algorithms is discussed. Finally, the future of LiDAR-based SLAM technology is examined.

Another very valuable read was [QGS15] which gives elaborate information on programming robots with ROS. It explains the small steps needed to set up a robot system which was greatly useful for this Master Thesis. An overview of ROS, an open-source robot operating system, is presented in [QCG+09] . ROS is not an operating system in the classic sense of process management and scheduling but rather a structured communications layer that sits on top of the host operating systems of a heterogeneous compute cluster as described by the paper. Also the website [soulee] is a true guide to deep dive into the world of ROS. It gives specific information to implement various functionalities on any robot model using ROS. [Zhe21] illustrates the various ways in which the ROS navigation stack parameters can be tuned. [RMV19] examines the Adaptive Monte Carlo Localization parameters (AMCL) against the results of the localization which helps in giving a closer understanding on the factors to be considered while setting the parameters for any given robot model. [XGG+19] performs SLAM using open-source algorithms gmapping, hector SLAM and karto SLAM to generate maps of the indoor environment and the concluded that the results are accurate and high precision grid maps can be constructed.

The paper [XZZH15] describes implementing an autonomous navigation system on wheeled mobile robots based on ROS. Robotic kinematics of motion is discussed for the custom robot's four-wheel differential drive mechanism. A new method of SLAM is introduced in the above-mentioned paper, which utilizes laser range sensors fixed in intelligent space. This approach utilized in the aforementioned research communicates with the mobile robot's SLAM. In addition to typical FastSLAM, the mobile robot's laser scan findings are compared with a map created by dispersed laser scanners in the intelligent space. As a result of the cooperative SLAM with the intelligent space, a more accurate map may be created. [HM11] introduces a new SLAM method, which utilizes laser range sensors fixed in intelligent space. This approach communicates with the mobile robot's SLAM. In addition to typical FastSLAM, the mobile robot's laser scan findings are compared with a map created by dispersed laser scanners in the intelligent space. A more accurate map may be created as a result of the cooperative SLAM with the intelligent space.

[MRM20] compares the travelled path and planned path of a virtual differential drive robot in Gazebo-ROS simulator and concluded that the differences mainly existed because of varied sensor accuracy, environment conditions, and number of obstacles. The evaluations are useful to improve the path planning and in turn the performance of the robot.[LS18] presents the design of a modular robot platform. The work involves the use of LiDAR and RGB-D camera kinect as the main sensors to obtain the information of the environment. Their experimental results show that the system can produce a map that is compatible with the environment in an indoor environment, and that the map can be used to accomplish the autonomous navigation task. A comparative study of five different SLAM algorithms is given in [JBM+19]. It gives a detailed description of the various metrics used to compare the algorithms and which one works best under the given circumstances.

All the above literature survey provided the author with the essential information to start the work and also allowed to understand the research journey from the fundamentals till date. It also helped to figure out the research methodology and reinforced the ideas to be used in the Master Thesis. All the unsolved problems presented as the drawbacks or limitations of their research work are potential goals for the Master's Thesis. It would have been impossible to proceed with the work without having a deep understanding of the background work done in the said field.

## 3.2 The Test Bed - Ohmni Telepresence Robot

The simulation model for the robot is developed by Ohmni labs to support the autonomy stack. At the moment the simulation includes :

- For RGB camera, KOLVN office model with texture and material. Sketchup was used to create this model, and Blender was used to add material and texture mapping.
- Sensor plugins: depth camera, navigation camera, telepresence camera, IMU, laser range sensor.
- Actuator plugins: differential drive, neck servo.

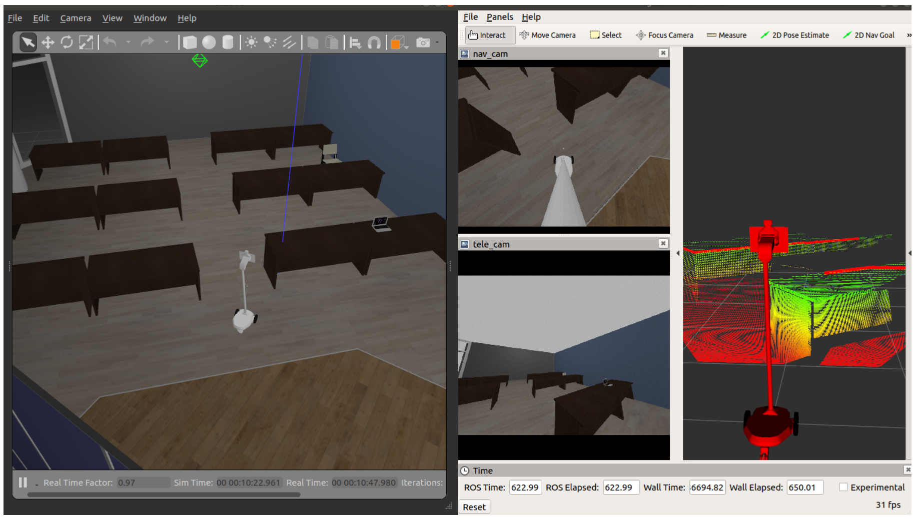


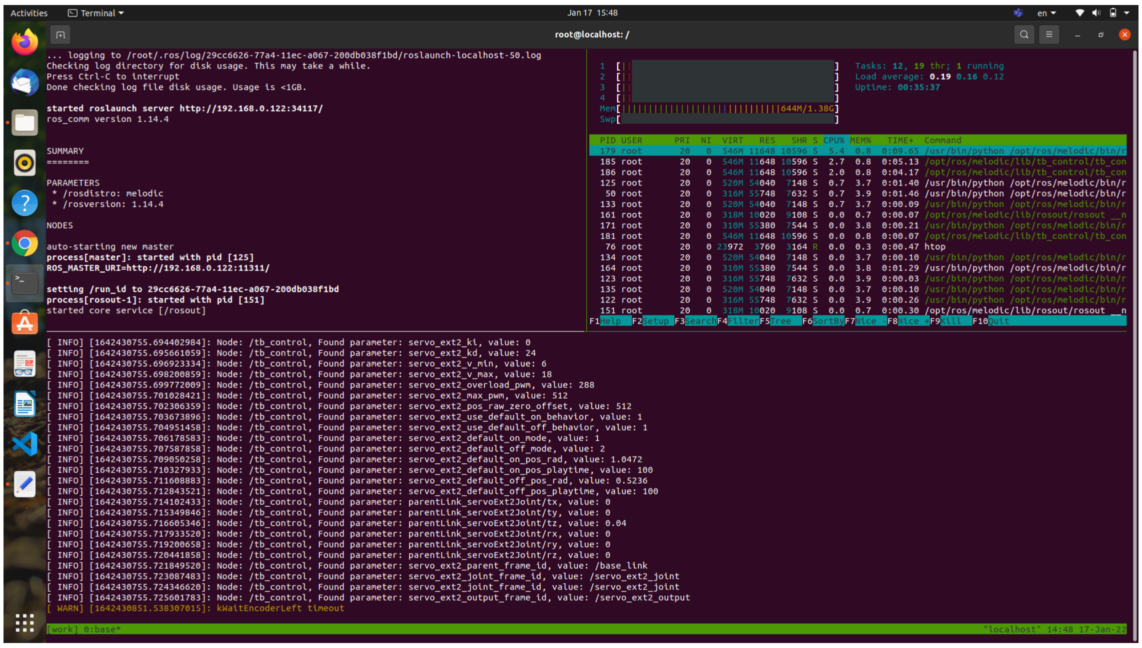
**Figure 3.1:** Gazebo model for Ohmni robot simulation

The simulation model is integrated with rviz and includes ROS packages like teleop to move the robot model in the simulation environment. Developers who want to use Ohmni in ROS or Gazebo with a simulation package can use the github repository to get the current public release of the Ohmni simulation models. The connection to the robot is required to start working with it or deploy custom code. Below are the steps to enable communication with the robot

- To access the shell on the smartphone, go to the Ohmni app's settings button (on the blue screen), and then tap 7 times on the "Version Name" item. This will reveal all hidden items.
- Scroll to the bottom and enable the "Enable Android Debug Bridge (ADB)" option. This will start the ADB daemon, allowing you to connect from the local network.

Once the connection is established, ROS developers can utilize the docker virtualization layer to run the ROS framework on top of Ohmni OS. Besides the tb-node which is required to operate the robot drivers. There is another node available as docker image _control to simply the task. The steps to use the docker image is as follows :

- It is crucial to turn off the serial port used by the tb-node, because the tb_control will also use the same serial port. The command to turn off the tb-node is :
  **In bot cli> setprop ctl.stop tb-node**
- The next step is to pull the docker image **In bot cli> docker pull"tb_control_docker_image"**
- Then run and access the docker. **In bot cli> docker run -it –network host –privileged -v /dev:/dev "tb_control_docker_image" bash**
- tb_control is running in a tmux session, could be accessed by the command **InDocker> tmux attach -t work**
- To check the connectivity run the command **rostopic list** on the local machine or inside the docker.

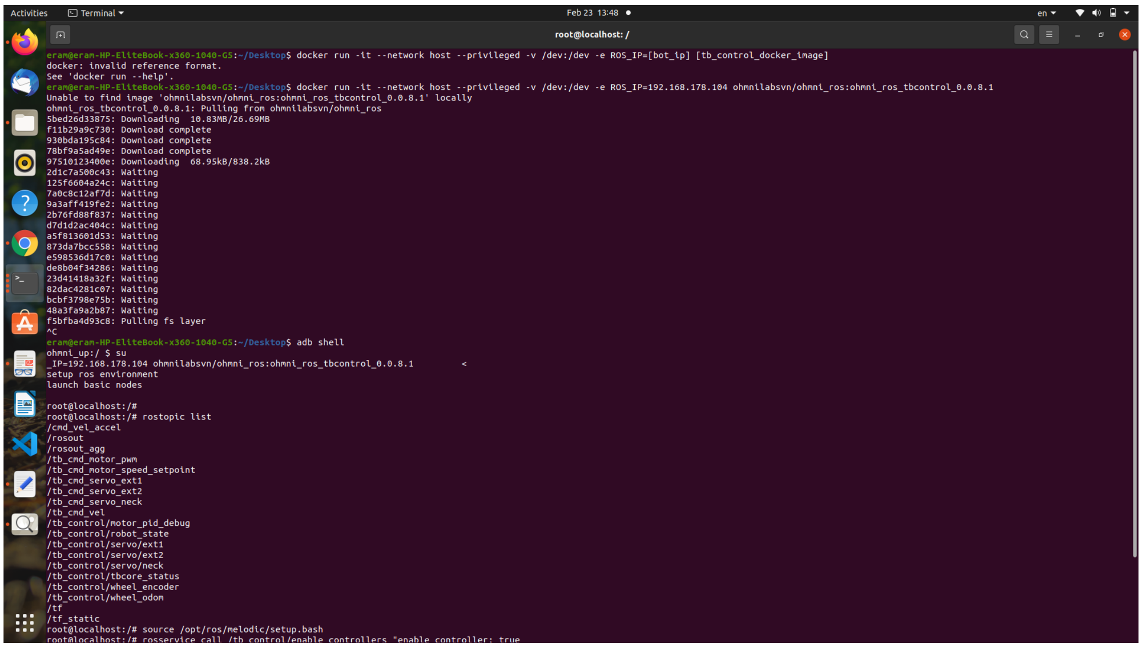


```
... logging to /root/.ros/log/29cc6626-77a4-11ec-a067-200db038f1bd/roslaunch-localhost-50.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.0.122:34117/
ros_comm version 1.14.4


SUMMARY
========

PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.4

NODES

auto-starting new master
process[master]: started with pid [125]
ROS_MASTER_URI=http://192.168.0.122:11311/

setting /run_id to 29cc6626-77a4-11ec-a067-200db038f1bd
process[rosout-1]: started with pid [151]
started core service [/rosout]
```

```
  1  [||                                    ]   Tasks: 12, 19 thr; 1 running
  2  [||                                    ]   Load average: 0.19 0.16 0.12
  3  [||                                    ]   Uptime: 00:35:37
  4  [||                                    ]
  Mem[||||||||||||||||||||||||||||644M/1.38G]
  Swp[                                      ]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
  179 root       20   0  546M 11648 10596 S  5.4  0.8  0:09.65 /usr/bin/python /opt/ros/melodic/bin/r
  185 root       20   0  546M 11648 10596 S  2.7  0.8  0:05.13 /opt/ros/melodic/lib/tb_control/tb_con
  186 root       20   0  546M 11648 10596 S  2.0  0.8  0:04.17 /opt/ros/melodic/lib/tb_control/tb_con
  125 root       20   0  520M 54040  7148 S  0.7  3.7  0:01.40 /usr/bin/python /opt/ros/melodic/bin/r
   50 root       20   0  316M 55748  7632 S  0.7  3.9  0:01.46 /usr/bin/python /opt/ros/melodic/bin/r
  133 root       20   0  520M 54040  7148 S  0.7  3.7  0:00.09 /usr/bin/python /opt/ros/melodic/bin/r
  161 root       20   0  318M 10020  9108 S  0.0  0.7  0:00.07 /opt/ros/melodic/lib/rosout/rosout __n
  171 root       20   0  310M 55380  7544 S  0.0  3.8  0:00.21 /usr/bin/python /opt/ros/melodic/bin/r
  181 root       20   0  546M 11648 10596 S  0.0  0.8  0:00.07 /opt/ros/melodic/lib/tb_control/tb_con
   76 root       20   0 23972  3760  3164 R  0.0  0.3  0:00.47 htop
  134 root       20   0  520M 54040  7148 S  0.0  3.7  0:00.10 /usr/bin/python /opt/ros/melodic/bin/r
  164 root       20   0  310M 55380  7544 S  0.0  3.8  0:01.29 /usr/bin/python /opt/ros/melodic/bin/r
  123 root       20   0  316M 55748  7632 S  0.0  3.9  0:00.03 /usr/bin/python /opt/ros/melodic/bin/r
  135 root       20   0  520M 54040  7148 S  0.0  3.7  0:00.10 /usr/bin/python /opt/ros/melodic/bin/r
  122 root       20   0  316M 55748  7632 S  0.0  3.9  0:00.26 /usr/bin/python /opt/ros/melodic/bin/r
  151 root       20   0  318M 10020  9108 S  0.0  0.7  0:00.30 /opt/ros/melodic/lib/rosout/rosout __n
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

```
[ INFO] [1642430755.694402984]: Node: /tb_control, Found parameter: servo_ext2_ki, value: 0
[ INFO] [1642430755.695661059]: Node: /tb_control, Found parameter: servo_ext2_kd, value: 24
[ INFO] [1642430755.696923334]: Node: /tb_control, Found parameter: servo_ext2_v_min, value: 6
[ INFO] [1642430755.698200859]: Node: /tb_control, Found parameter: servo_ext2_v_max, value: 18
[ INFO] [1642430755.699772009]: Node: /tb_control, Found parameter: servo_ext2_overload_pwm, value: 288
[ INFO] [1642430755.701028421]: Node: /tb_control, Found parameter: servo_ext2_max_pwm, value: 512
[ INFO] [1642430755.702306359]: Node: /tb_control, Found parameter: servo_ext2_pos_raw_zero_offset, value: 512
[ INFO] [1642430755.703673896]: Node: /tb_control, Found parameter: servo_ext2_use_default_on_behavior, value: 1
[ INFO] [1642430755.704951458]: Node: /tb_control, Found parameter: servo_ext2_use_default_off_behavior, value: 1
[ INFO] [1642430755.706178583]: Node: /tb_control, Found parameter: servo_ext2_default_on_mode, value: 1
[ INFO] [1642430755.707587858]: Node: /tb_control, Found parameter: servo_ext2_default_off_mode, value: 2
[ INFO] [1642430755.709050258]: Node: /tb_control, Found parameter: servo_ext2_default_on_pos_rad, value: 1.0472
[ INFO] [1642430755.710327933]: Node: /tb_control, Found parameter: servo_ext2_default_on_pos_playtime, value: 100
[ INFO] [1642430755.711608883]: Node: /tb_control, Found parameter: servo_ext2_default_off_pos_rad, value: 0.5236
[ INFO] [1642430755.712843521]: Node: /tb_control, Found parameter: servo_ext2_default_off_pos_playtime, value: 100
[ INFO] [1642430755.714102433]: Node: /tb_control, Found parameter: parentLink_servoExt2Joint/tx, value: 0
[ INFO] [1642430755.715349846]: Node: /tb_control, Found parameter: parentLink_servoExt2Joint/ty, value: 0
[ INFO] [1642430755.716605346]: Node: /tb_control, Found parameter: parentLink_servoExt2Joint/tz, value: 0.04
[ INFO] [1642430755.717933520]: Node: /tb_control, Found parameter: parentLink_servoExt2Joint/rx, value: 0
[ INFO] [1642430755.719200658]: Node: /tb_control, Found parameter: parentLink_servoExt2Joint/ry, value: 0
[ INFO] [1642430755.720441858]: Node: /tb_control, Found parameter: parentLink_servoExt2Joint/rz, value: 0
[ INFO] [1642430755.721849520]: Node: /tb_control, Found parameter: servo_ext2_parent_frame_id, value: /base_link
[ INFO] [1642430755.723087483]: Node: /tb_control, Found parameter: servo_ext2_joint_frame_id, value: /servo_ext2_joint
[ INFO] [1642430755.724346620]: Node: /tb_control, Found parameter: servo_ext2_joint_frame_id, value: /servo_ext2_joint
[ INFO] [1642430755.725601783]: Node: /tb_control, Found parameter: servo_ext2_output_frame_id, value: /servo_ext2_output
[ WARN] [1642430851.538307015]: kWaitEncoderLeft timeout

[work] 0:base*                                                                          "localhost" 14:48 17-Jan-22
```

**Figure 3.2:** The tb_control node running in a tmux session after running the above command



```
eram@eram-HP-EliteBook-x360-1040-G5:~/Desktop$ docker run -it --network host --privileged -v /dev:/dev -e ROS_IP=[bot_ip] [tb_control_docker_image]
docker: invalid reference format.
See 'docker run --help'.
eram@eram-HP-EliteBook-x360-1040-G5:~/Desktop$ docker run -it --network host --privileged -v /dev:/dev -e ROS_IP=192.168.178.104 ohmnilabsvn/ohmni_ros:ohmni_ros_tbcontrol_0.0.8.1
Unable to find image 'ohmnilabsvn/ohmni_ros:ohmni_ros_tbcontrol_0.0.8.1' locally
ohmni_ros_tbcontrol_0.0.8.1: Pulling from ohmnilabsvn/ohmni_ros
5bed26d33875: Downloading  10.83MB/26.69MB
f11b29a9c730: Download complete
930bda195c84: Download complete
78bf9a5ad49e: Download complete
97510123400e: Downloading  68.95kB/838.2kB
2d1c7a500c43: Waiting
125f6604a24c: Waiting
7a0c8c12af7d: Waiting
9a3aff419fe2: Waiting
2b76fd88f837: Waiting
d7d1d2ac404c: Waiting
a5f813601d53: Waiting
873da7bcc558: Waiting
e598536d17c0: Waiting
de8b04f34286: Waiting
23d41418a32f: Waiting
82dac4281c07: Waiting
bcbf3798e75b: Waiting
48a3fa9a2b87: Waiting
f5bfba4d93c8: Pulling fs layer
^C
eram@eram-HP-EliteBook-x360-1040-G5:~/Desktop$ adb shell
ohmni_up:/ $ su
_IP=192.168.178.104 ohmnilabsvn/ohmni_ros:ohmni_ros_tbcontrol_0.0.8.1         <
setup ros environment
launch basic nodes

root@localhost:/#
root@localhost:/# rostopic list
/cmd_vel_accel
/rosout
/rosout_agg
/tb_cmd_motor_pwm
/tb_cmd_motor_speed_setpoint
/tb_cmd_servo_ext1
/tb_cmd_servo_ext2
/tb_cmd_servo_neck
/tb_cmd_vel
/tb_control/motor_pid_debug
/tb_control/robot_state
/tb_control/servo/ext1
/tb_control/servo/ext2
/tb_control/servo/neck
/tb_control/tbcore_status
/tb_control/wheel_encoder
/tb_control/wheel_odom
/tf
/tf_static
root@localhost:/# source /opt/ros/melodic/setup.bash
root@localhost:/# rosservice call /tb_control/enable_controllers "enable_controller: true
```

**Figure 3.3:** Set of commands ran on the Ubuntu Linux to enable connection with the Ohmni Telepresence Robot and checking the connection using the command rostopic list.

# 4 Solution Design

## 4.1 System Components

### 4.1.1 Hardware Setup

#### 4.1.1.1 LiDAR - Light Detection and Ranging

One of the most important devices for the Thesis was a laser sensor to perform SLAM and obstacle detection. After doing a bit of research it was decided to use a YDLIDAR X4. It's a low-cost 2D lidar with a 360-degree range (99 dollars). Only a few lidars are as inexpensive as this one, one of which is the RPLidar A1M8. One of the reasons for choosing this LiDAR is that it is compatible with the ROS and the Ohmni robot. Connect the adapter board and the X4 first, then the Universal Serial Bus (USB) cord to the adaptor board's USB port and the Personal Computer (PC). The Micro interface of the USB interface is connected to the USB DATA of the USB adapter. The X4 enters idle mode after being switched on, and the motor does not rotate. Because the drive current of some development platforms or PCs' USB interfaces is low, the X4 requires access to the +5V auxiliary power source; otherwise, the radar will operate improperly. In that instance, the USB-PWR pin can be used to provide power. When the ydlidar was attached to an HP laptop, only the USB DATA point was required, but when it was linked to a Raspberry Pi, both the data and power pins were required.

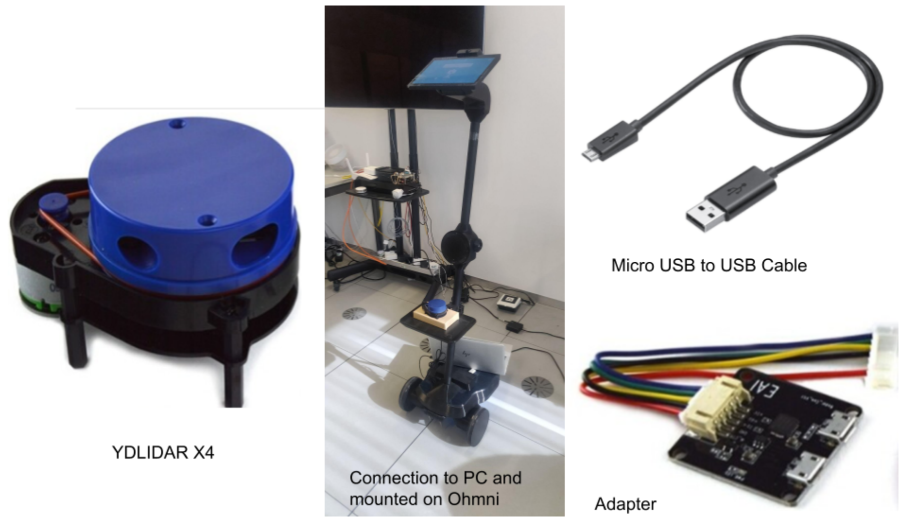


**Figure 4.1:** YdLiDAR X4 mounted on Ohmni Robot and connection made to the PC

#### 4.1.1.2 Raspberry Pi

The Raspberry Pi is a small device that can run complete Linux operating systems. Raspbian is an operating system for the Raspberry Pi that is based on Debian (a Linux distribution). Ubuntu Mate is installed on the Pi in this project. It supports ROS which has been discussed in the previous chapter. The Raspberry Pi was first released in 2011 and has seen numerous upgrades since then. The Raspberry Pi 4 model B+, with a 1.5 GHz processor, four cores, four USB ports, and an uncoated circuit board, is used. The Raspberry Pi also has an Synchronous Dynamic Random Access Memory (SDRAM) of 8 GB and a micro-SD socket for a bootable micro-SD. To control the Raspberry Pi from the laptop i.e. to send the commands and launch the visualization, Virtual Network Computing (VNC) connector was employed. The block diagram gives the details of the data flow between the different hardware components used.

#### 4.1.1.3 Power Bank

A power bank to supply constant power to the Raspberry Pi is used. It is a 20100 mAh USB-C type product by Aukey. It ensures that the device is charged at the fastest rate upto 2.4 A. The power bank was needed as we require the robot to move around freely in the lab and having a wireless power supply was the only option.

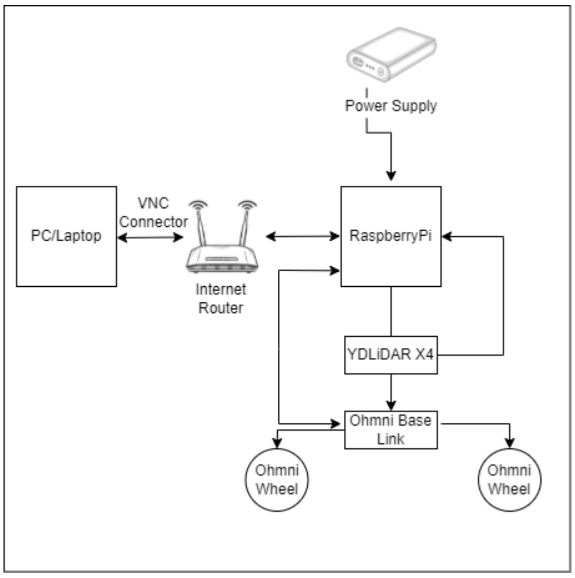


**Figure 4.2:** Hardware architecture and data flow

### 4.1.2 Software Setup

#### 4.1.2.1 Hector SLAM Mapping

Simultaneous Localization and Mapping (SLAM) is a problem that attempts to localize itself while also mapping the environment. Depending on the situation, there are a variety of options. The SLAM system is a collection of algorithms that work together to tackle challenges like simulation and simultaneous mapping. As previously stated, this project makes use of ROS (Robot Operating System), a framework for controlling robotic components from a computer. The ROS system is made up of a number of free nodes that communicate with each other via a publisher/subscribe model. The Robot Operating System (ROS) is a software development framework for robots. It is a set of tools, libraries, and protocols designed at making the work of developing strong and resilient robot behavior easier across a variety of robot platforms. ROS was created from the ground up to encourage the development of robotic collaborative software.

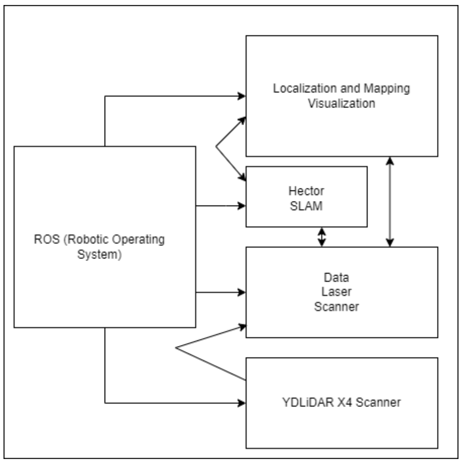

Localization and Mapping Visualization
Hector SLAM
ROS (Robotic Operating System)
Data Laser Scanner
YDLiDAR X4 Scanner

**Figure 4.3:** Basic components of software system

The performance of HectorSLAM was examined in this study. HectorSLAM combines a powerful scan matching technique with a 2D SLAM system. In this experiment, the estimation of robot movement in real time is put to the test. For particle filter-based mapping, Hector SLAM is considered state of the art. This SLAM technique can be utilized without an odometer and on platforms that display sensor roll or pitch motion. While the system lacks explicit loop closing capability, it is accurate enough for many real-world applications. So, the first step is to install the HectorSLAM package for ROS and add the launch file shown in **Figure 4.4**. This file is a supplement to the lidar.launch file, which can be found in the launch folder. To do mapping, a few things are added to the lidar.launch file. In addition to the base link to laser broadcaster, two transform broadcasters have been introduced.

1. map to odom
2. odom to base_link

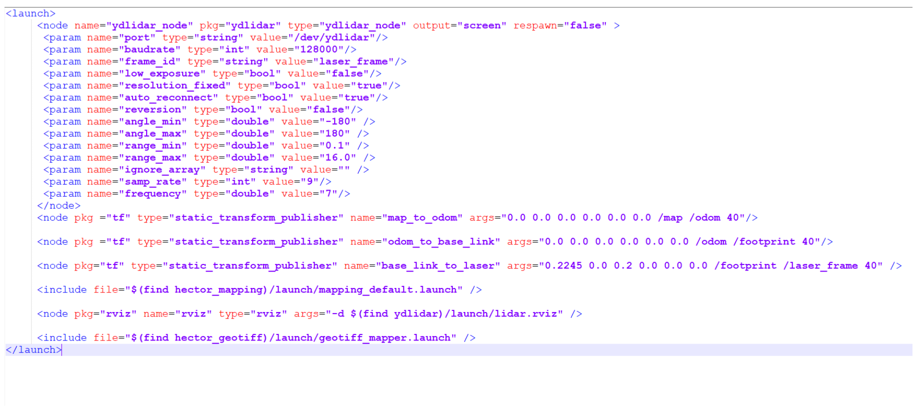
There are two launch files supplied.

1. mapping_default from hector mapping package
2. From the hector geotiff package, geotiff_mapper.launch.

The parameter values from the previous file, lidar.launch from the ydlidar package have been preserved. The transform broadcaster was required to transform data from the laser node to the map node; otherwise, the map would not be valid.

#### 4.1.2.2 Robot Setup - Location of sensors and coordinate frames

The location and placement of sensors on the robot are critical since they determine perception and measurement accuracy. The type of measurement and the sensor's range determine the sensor's location. The location of LiDAR and its transform frames with regard to the ohmni robot base



```
<launch>
    <node name="ydlidar_node" pkg="ydlidar" type="ydlidar_node" output="screen" respawn="false" >
     <param name="port" type="string" value="/dev/ydlidar"/>
     <param name="baudrate" type="int" value="128000"/>
     <param name="frame_id" type="string" value="laser_frame"/>
     <param name="low_exposure" type="bool" value="false"/>
     <param name="resolution_fixed" type="bool" value="true"/>
     <param name="auto_reconnect" type="bool" value="true"/>
     <param name="reversion" type="bool" value="false"/>
     <param name="angle_min" type="double" value="-180" />
     <param name="angle_max" type="double" value="180" />
     <param name="range_min" type="double" value="0.1" />
     <param name="range_max" type="double" value="16.0" />
     <param name="ignore_array" type="string" value="" />
     <param name="samp_rate" type="int" value="9"/>
     <param name="frequency" type="double" value="7"/>
    </node>
    <node pkg ="tf" type="static_transform_publisher" name="map_to_odom" args="0.0 0.0 0.0 0.0 0.0 0.0 /map /odom 40"/>

    <node pkg ="tf" type="static_transform_publisher" name="odom_to_base_link" args="0.0 0.0 0.0 0.0 0.0 0.0 /odom /footprint 40"/>

    <node pkg="tf" type="static_transform_publisher" name="base_link_to_laser" args="0.2245 0.0 0.2 0.0 0.0 0.0 /footprint /laser_frame 40" />

    <include file="$(find hector_mapping)/launch/mapping_default.launch" />

    <node pkg="rviz" name="rviz" type="rviz" args="-d $(find ydlidar)/launch/lidar.rviz" />

    <include file="$(find hector_geotiff)/launch/geotiff_mapper.launch" />
</launch>
```

**Figure 4.4:** Launch file for HectorSLAM node

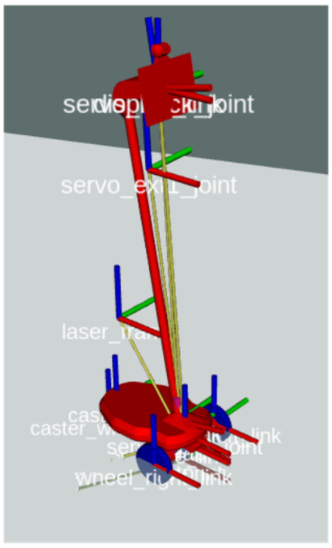
will be discussed in the next subsections. The traditional method of determining the position and orientation of a robot's **link** (component/part of the robot) is to assign a coordinate to it. A **joint** connects two or more links together.

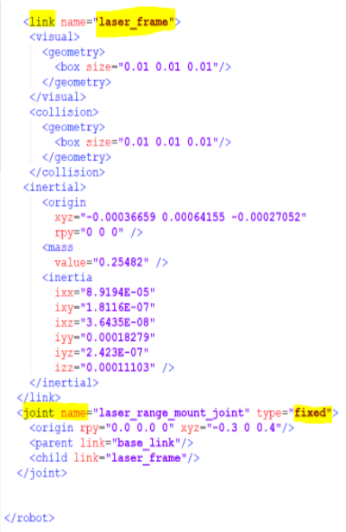
With the use of transformation matrices, the mobility of each link with respect to the robot base (or base link) may be simply computed once the coordinate frames for the system have been defined. A transformation matrix is a helpful tool made up of combinations of rotation and translation matrices. An example of a transformation matrix is shown below, which is defined as a transition from arbitrary frame A to arbitrary frame B.

$$^{A}T_{B} = \left(\begin{array}{ccc|c} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$$

**Figure 4.5:** Transition from arbitraty frame A to arbitraty frame B

The Unified Robotic Description Format (URDF) is an Extensible Markup Language (XML) file format for describing all aspects of a robot in ROS. The robot model and the coordinate frames were created using ROS URDF package to match the actual robot description. The coordinate frames and their names are shown in Fig. 4.6 with the robot model. These coordinate frames are used to establish transformations between various sections of the robot, which aids in defining the mobility of specific links (parts) in relation to the robot base frame and the world frame.
**LiDAR (laser frame)** The LiDAR is being utilized in this project to create a 2D map of the robot's surroundings, with special attention paid to the walls of the room or laboratory, as well as obstacles such as tables and chairs. The LiDAR could not be mounted at the bottom of the robot since it would negate the aim of identifying objects that are 50-100 cm off the ground. A standard office chair, for example, has four wheels and is supported by a rod-like framework; if the laser is placed





```
<link name="laser_frame">
  <visual>
    <geometry>
      <box size="0.01 0.01 0.01"/>
    </geometry>
  </visual>
  <collision>
    <geometry>
      <box size="0.01 0.01 0.01"/>
    </geometry>
  </collision>
 <inertial>
    <origin
      xyz="-0.00036659 0.00064155 -0.00027052"
      rpy="0 0 0" />
    <mass
      value="0.25482" />
    <inertia
      ixx="8.9194E-05"
      ixy="1.8116E-07"
      ixz="3.6435E-08"
      iyy="0.00018279"
      iyz="2.423E-07"
      izz="0.00011103" />
  </inertial>
</link>
<joint name="laser_range_mount_joint" type="fixed">
  <origin rpy="0.0 0.0 0" xyz="-0.3 0 0.4"/>
  <parent link="base_link"/>
  <child link="laser_frame"/>
</joint>


</robot>
```

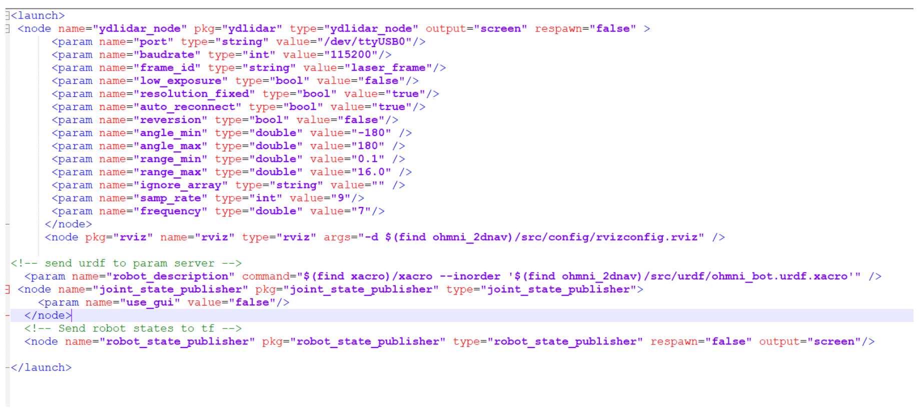
**Figure 4.6:** Coordinate frames of the Ohmni robot with the laser_frame integrated and the corresponding code from the urdf file with the laser frame link and joint

at a low position, it will only identify the support as a point rather than the whole body of the chair, resulting in false data on the map. Similarly, the LiDAR could not be mounted too high on the robot since it would miss any nearby obstacles. The trade-off determined that the LiDAR should be mounted on the robot's tube link at around 50 cm above ground level, just below the average height of obstacles in the environment.

#### 4.1.2.3 Robot Configuration Setup

**Robot Configuration Launch file :** The robot uses tf to publish coordinate frame information, receiving sensor msgs/LaserScan messages from the LiDAR that will be used with the navigation stack, and publishing odometry data using both tf and the nav msgs/Odometry message, as well as receiving velocity commands to send to the base.
The first step was to create a package that contained all of the navigation stack's configuration and launch files. This package depends on the move base package as well as any other packages used to create the Robot URDF model above. The next step was to create a roslaunch file that brings together the hardware and transform publishes that the robot needs. The robot URDF file is called in this launch file. Any sensors that the robot uses for navigation is brought up in this step. The laser sensor was used in this work. This file also contains the odometry for the robot's base as well as the transform configuration. The file can be viewed in the figure 4.7 given below.

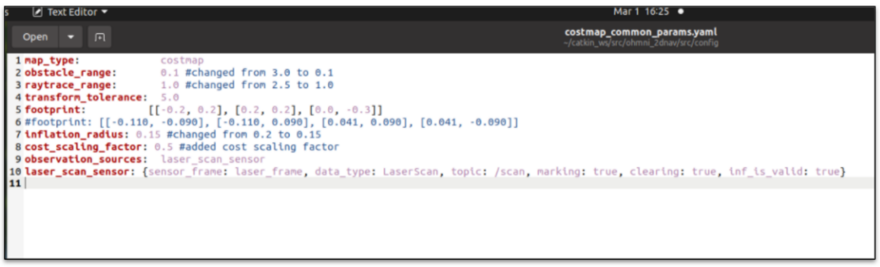
**Costmap Configuration:** Costmaps are used by the navigation stack to hold information about obstacles in the real world. The costmaps need to be aware of the sensor topics they should listen to for changes in order to achieve this successfully. There are two kinds of costmaps - local and global. There are some parameters which are common to both costmaps and therefore listed under a common file - common_costmap.



```xml
<launch>
 <node name="ydlidar_node" pkg="ydlidar" type="ydlidar_node" output="screen" respawn="false" >
      <param name="port" type="string" value="/dev/ttyUSB0"/>
      <param name="baudrate" type="int" value="115200"/>
      <param name="frame_id" type="string" value="laser_frame"/>
      <param name="low_exposure" type="bool" value="false"/>
      <param name="resolution_fixed" type="bool" value="true"/>
      <param name="auto_reconnect" type="bool" value="true"/>
      <param name="reversion" type="bool" value="false"/>
      <param name="angle_min" type="double" value="-180" />
      <param name="angle_max" type="double" value="180" />
      <param name="range_min" type="double" value="0.1" />
      <param name="range_max" type="double" value="16.0" />
      <param name="ignore_array" type="string" value="" />
      <param name="samp_rate" type="int" value="9"/>
      <param name="frequency" type="double" value="7"/>
     </node>
     <node pkg="rviz" name="rviz" type="rviz" args="-d $(find ohmni_2dnav)/src/config/rvizconfig.rviz" />

<!-- send urdf to param server -->
  <param name="robot_description" command="$(find xacro)/xacro --inorder '$(find ohmni_2dnav)/src/urdf/ohmni_bot.urdf.xacro'" />
 <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher">
    <param name="use_gui" value="false"/>
  </node>
  <!-- Send robot states to tf -->
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" respawn="false" output="screen"/>

</launch>
```

**Figure 4.7:** Ohmni Configuration Launch file

The **common_costmap_params file** contains parameters that set threshold limits on obstacle information put into costmap is found in common costmap params file. The "obstacle range" parameter specifies the maximum sensor range at which an obstacle will be included to the costmap. It is set to 0.1 meters, which means the robot's map will only be updated with information about obstacles within 0.1 meters of the base. The "raytrace range" parameter specifies the range for which we will raytrace freespace for a given sensor reading. Setting it to 1.0 meters indicates that, given a sensor reading, the robot will attempt to clear space in front of it up to 1.0 meters away. For small spaces, it is nice to keep the value in centimeters as very high value could lead to high cost which might lead to inaccurate results. The footprint of the robot is rough estimate of the dimension of robot base. Setting the inflating radius to 0.2 meters, indicates that all paths that stay 0.2 meters or more away from obstacles will be treated as having the same obstacle cost. The "observation sources" parameter specifies a laser sensor that will send data to the costmap. The **global_costmap_param file** contains "global frame" which specifies the coordinate frame in


Text Editor
Mar 1 16:25
Open
costmap_common_params.yaml
~/catkin_ws/src/ohmni_2dnav/src/config

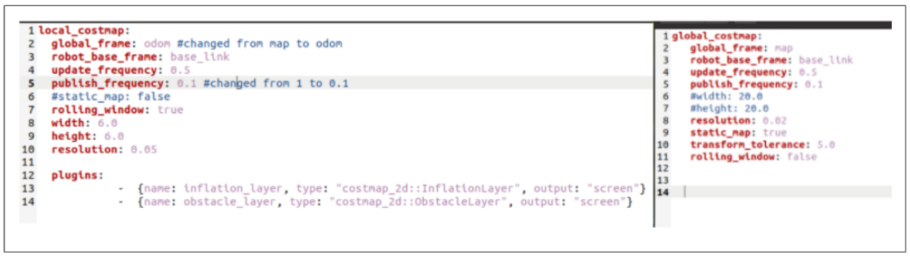
```yaml
map_type:             costmap
obstacle_range:       0.1 #changed from 3.0 to 0.1
raytrace_range:       1.0 #changed from 2.5 to 1.0
transform_tolerance:  5.0
footprint:          [[-0.2, 0.2], [0.2, 0.2], [0.0, -0.3]]
#footprint: [[-0.110, -0.090], [-0.110, 0.090], [0.041, 0.090], [0.041, -0.090]]
inflation_radius: 0.15 #changed from 0.2 to 0.15
cost_scaling_factor: 0.5 #added cost scaling factor
observation_sources:  laser_scan_sensor
laser_scan_sensor: {sensor_frame: laser_frame, data_type: LaserScan, topic: /scan, marking: true, clearing: true, inf_is_valid: true}
```

**Figure 4.8:** common_costmap_params.yaml file

which the costmap should execute; the /map frame in this project. The "robot base frame" specifies the coordinate frame used by the costmap for the robot's base. The "update frequency" sets the frequency at which the costmap's update loop will run in Hz. The update frequency and publish frequency are set to **low values** because it affects the performance of the code if set otherwise. The parameters were tuned to suit the Ohmni robot system. Setting the frequency to a high value would put a lot of pressure on the RAM of the controller in this case the Raspberry Pi. The "static map" states whether the costmap should be initialized using a map from the map server. It is set to true because an already saved map is being served to it.

The “costmap 2d::Costmap2DROS” object makes considerable use of tf to insert data from sensor sources into the costmap. It assumes that all transforms between the global frame, robot base frame, and sensor sources are connected and up-to-date. The **transform tolerance** parameter specifies the maximum amount of latency between these transforms that can be tolerated. The navigation stack will stop the robot if the tf tree does not update at the expected rate. The parameter “rolling window” implies that the costmap does not represent the entire environment, but merely the immediate surrounding (e.g. 5m x 5m around your robot). The costmap will then follow the robot and be updated when new sensor data is received. Therefore it is set to false in the global map.

**The local_costmap parameters file** specifies some plugins. In two dimensions, the Obstacle-CostmapPlugin marks and raytraces obstacles. The inflating layer is a costmap optimization that adds extra values around lethal barriers (i.e. inflates the obstacles) to represent the robot’s configuration space. Other parameters are same as the global costmap.



```
local_costmap:
  global_frame: odom #changed from map to odom
  robot_base_frame: base_link
  update_frequency: 0.5
  publish_frequency: 0.1 #changed from 1 to 0.1
  #static_map: false
  rolling_window: true
  width: 6.0
  height: 6.0
  resolution: 0.05

  plugins:
            - {name: inflation_layer, type: "costmap_2d::InflationLayer", output: "screen"}
            - {name: obstacle_layer, type: "costmap_2d::ObstacleLayer", output: "screen"}
```

```
global_costmap:
    global_frame: map
    robot_base_frame: base_link
    update_frequency: 0.5
    publish_frequency: 0.1
    #width: 20.0
    #height: 20.0
    resolution: 0.02
    static_map: true
    transform_tolerance: 5.0
    rolling_window: false
```

**Figure 4.9:** The local and global parameters costmap files

**Base Local Planner Configuration:** Given a high-level plan, the move base local planner is responsible for determining velocity commands to send to the robot’s mobile base. The move_base_params and dwa_local_planner together set the parameters in this case. The **Figure 4.10** gives a detail of the values for these parameters. The final velocity of the robot is set to 0.5 m/s as it was calculated to be the optimum.

#### 4.1.2.4 Navigation Stack Setup

**AMCL Package Configuration:** The AMCL package keeps track of a probability distribution for all conceivable robot poses and updates it using data from odometry and laser rangefinders. A predetermined map of the environment against which to compare observed sensor data is also required by the package. The AMCL package uses a particle filter to describe the probability distribution at the implementation level. The filter is “adaptive” because the number of particles in the filter is dynamically adjusted: when the robot’s pose is very uncertain, the number of particles is increased; when the robot’s pose is well known, the number of particles is reduced. This leads to a trade-off between the processing speed of the robot and the localization accuracy.

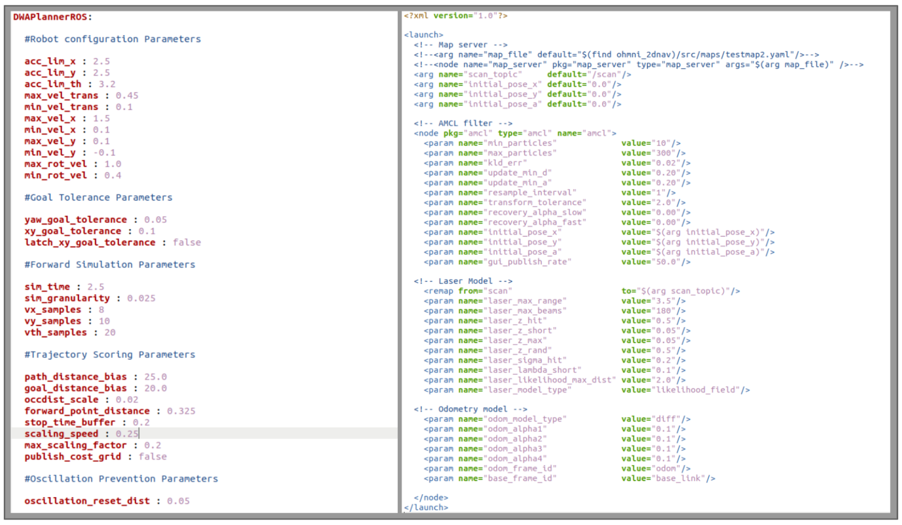
There are a number of parameters that can be tweaked based on one’s understanding of the platform and sensors employed. Configuring these parameters can improve the AMCL package’s performance and accuracy while reducing the number of recovery rotations the robot makes while navigating. Overall filter, laser model, and odometry model are the three types of ROS parameters that can be utilized to configure the AMCL node.

| Subscribed Topics (message type) | Published Topics (message type) |
|---|---|
| map ( nav_msgs/OccupancyGrid) | amcl_pose (geometry_msgs/PoseWithCovarianceStamped) |
| scan (sensor_msgs/LaserScan) | particlecloud (geometry_msgs/PoseArray) |
| tf (tf/tfMessage) | tf (tf/tfMessage) |

**Table 4.1:** List of Subscribed and Published Topics by AMCL

- **map:** AMCL subscribes to the map topic to obtain map data Occupancy Grid Mapping (OGM) for use in localization.
- **scan:** To obtain the most recent scan data.
- **tf:** Transform topic, which is required to provide the relationship between various reference frames. For example, translate from the base laser (laser reference frame) coordinate frame to the base link (robot center) coordinate frame.
- **amcl pose:** The amcl node communicates the robot's position in the environment to the amcl_pose topic. AMCL publishes the particle cloud of arrows generated by the system to measure the uncertainty of the robot's present position. Red arrows displayed using Rviz by adding the **PoseArray** display which subscribes to PointCloud topic.

The figure below shows a complete list of these setup settings, as well as additional information about the package.



```
DWAPlannerROS:

  #Robot configuration Parameters

  acc_lim_x : 2.5
  acc_lim_y : 2.5
  acc_lim_th : 3.2
  max_vel_trans : 0.45
  min_vel_trans : 0.1
  max_vel_x : 1.5
  min_vel_x : 0.1
  max_vel_y : 0.1
  min_vel_y : -0.1
  max_rot_vel : 1.0
  min_rot_vel : 0.4

  #Goal Tolerance Parameters

  yaw_goal_tolerance : 0.05
  xy_goal_tolerance : 0.1
  latch_xy_goal_tolerance : false

  #Forward Simulation Parameters

  sim_time : 2.5
  sim_granularity : 0.025
  vx_samples : 8
  vy_samples : 10
  vth_samples : 20

  #Trajectory Scoring Parameters

  path_distance_bias : 25.0
  goal_distance_bias : 20.0
  occdist_scale : 0.02
  forward_point_distance : 0.325
  stop_time_buffer : 0.2
  scaling_speed : 0.25
  max_scaling_factor : 0.2
  publish_cost_grid : false

  #Oscillation Prevention Parameters

  oscillation_reset_dist : 0.05
```

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

<launch>
  <!-- Map server -->
  <!--<arg name="map_file" default="$(find ohmni_2dnav)/src/maps/testmap2.yaml"/>-->
  <!--<node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)" />-->
  <arg name="scan_topic"     default="/scan"/>
  <arg name="initial_pose_x" default="0.0"/>
  <arg name="initial_pose_y" default="0.0"/>
  <arg name="initial_pose_a" default="0.0"/>

  <!-- AMCL filter -->
  <node pkg="amcl" type="amcl" name="amcl">
    <param name="min_particles"             value="10"/>
    <param name="max_particles"             value="300"/>
    <param name="kld_err"                   value="0.02"/>
    <param name="update_min_d"              value="0.20"/>
    <param name="update_min_a"              value="0.20"/>
    <param name="resample_interval"         value="1"/>
    <param name="transform_tolerance"       value="2.0"/>
    <param name="recovery_alpha_slow"       value="0.00"/>
    <param name="recovery_alpha_fast"       value="0.00"/>
    <param name="initial_pose_x"            value="$(arg initial_pose_x)"/>
    <param name="initial_pose_y"            value="$(arg initial_pose_y)"/>
    <param name="initial_pose_a"            value="$(arg initial_pose_a)"/>
    <param name="gui_publish_rate"          value="50.0"/>

  <!-- Laser Model -->
    <remap from="scan"                      to="$(arg scan_topic)"/>
    <param name="laser_max_range"           value="3.5"/>
    <param name="laser_max_beams"           value="180"/>
    <param name="laser_z_hit"               value="0.5"/>
    <param name="laser_z_short"             value="0.05"/>
    <param name="laser_z_max"               value="0.05"/>
    <param name="laser_z_rand"              value="0.5"/>
    <param name="laser_sigma_hit"           value="0.2"/>
    <param name="laser_lambda_short"        value="0.1"/>
    <param name="laser_likelihood_max_dist" value="2.0"/>
    <param name="laser_model_type"          value="likelihood_field"/>

  <!-- Odometry model -->
    <param name="odom_model_type"           value="diff"/>
    <param name="odom_alpha1"               value="0.1"/>
    <param name="odom_alpha2"               value="0.1"/>
    <param name="odom_alpha3"               value="0.1"/>
    <param name="odom_alpha4"               value="0.1"/>
    <param name="odom_frame_id"             value="odom"/>
    <param name="base_frame_id"             value="base_link"/>

  </node>
</launch>
```

**Figure 4.10:** (left) DWA Local Planner config file and (right) startLocalization launch file with the tuned parameters for improved localization

**The move base node of the ROS Navigation Stack** uses the configuration files. Behind the scenes, the move base node is in charge of planning a collision-free path for a mobile robot from a beginning point to a goal location. The launch file launches the map, amcl node and the config files. Once the launch is complete, the transforms take place and odom data is received by the move base server. This completes the navigation stack set up for the robot. The frames and rosgraph can be viewed that depicts the communication between the various nodes and the messages share between them.

#### 4.1.2.5 Map-Based Autonomous Navigation

Following the creation of a map and the localization of the robot, a trajectory for the robot must be created to follow in order to reach a certain goal while avoiding obstacles along the way. To accomplish this, the move base package is used as mentioned in the previous chapter, which includes the move base node. This package serves as the foundation of the navigation task, connecting all of the navigation components. With a message of type **geometry msgs/PoseStamped**, the move base node implements **SimpleActionServer** from the **actionlib** package. The action server provides the **/goal** topic, which provides the move base node with the goal point. See the table below for a description of some of the topics to which the move base node subscribes and publishes.

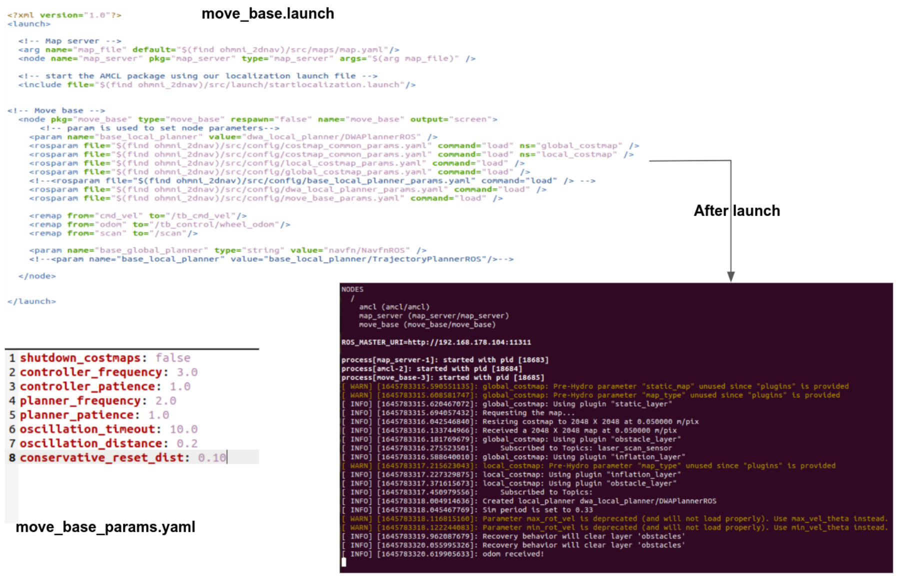
| **Topics** | **Messages** | **Description** |
|---|---|---|
| move_base/goal (subscribed) | move_base_msgs/MoveBaseActionGoal | Provide goal using SimpleActionServer which will allow tracking of current goal position status. |
| move_base_simple/goal ( subscribed) | geometry_msgs/PoseStamped | Provide goal position to /move_base node directly without using SimpleActionServer. |
| /cmd_vel (published) | geometry_msgs/Twist | Publish velocity data to the robot (base controller to make transformation). |

**Table 4.2:** Topics move_base subscribes and publishes

To create a goal, one can use Rviz.

- The map is generated, and the robot is localized.
- In the launch file an rviz.config file is included which opens rviz automatically with the necessary displays.
- To see the local and global costmaps, two Map display elements are added to the /move_base/local costmap/costmap and /move_base/global costmap/costmap topics.
- Two Path display elements are added and linked to the /move_base/DWAPlannerROS/local plan and /move_base/NavfnROS/plan topics, respectively, to visualize the local and global plans.
- Select the 2D Nav Goal button and point the robot to the location where it should move (goal). on the map. Following the creation of a goal, a goal message (geometry_msgs/PoseStamped) is broadcast to the /move_base/goal topic.

- Another method for creating our goal is to write an action client program (node) that sends a goal to move base shown in the figure below.

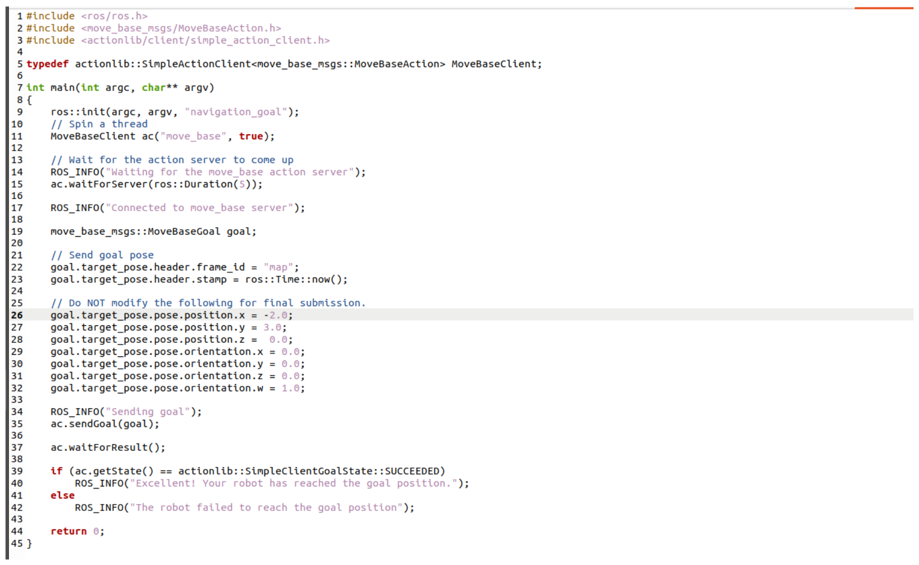


```xml
<?xml version="1.0"?>
<launch>

  <!-- Map server -->
  <arg name="map_file" default="$(find ohmni_2dnav)/src/maps/map.yaml"/>
  <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)" />

  <!-- start the AMCL package using our localization launch file -->
  <include file="$(find ohmni_2dnav)/src/launch/startlocalization.launch"/>


<!-- Move base -->
  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
      <!-- param is used to set node parameters-->
    <param name="base_local_planner" value="dwa_local_planner/DWAPlannerROS" />
    <rosparam file="$(find ohmni_2dnav)/src/config/costmap_common_params.yaml" command="load" ns="global_costmap" />
    <rosparam file="$(find ohmni_2dnav)/src/config/costmap_common_params.yaml" command="load" ns="local_costmap" />
    <rosparam file="$(find ohmni_2dnav)/src/config/local_costmap_params.yaml" command="load" />
    <rosparam file="$(find ohmni_2dnav)/src/config/global_costmap_params.yaml" command="load" />
    <!--<rosparam file="$(find ohmni_2dnav)/src/config/base_local_planner_params.yaml" command="load" /> -->
    <rosparam file="$(find ohmni_2dnav)/src/config/dwa_local_planner_params.yaml" command="load" />
    <rosparam file="$(find ohmni_2dnav)/src/config/move_base_params.yaml" command="load" />

    <remap from="cmd_vel" to="/tb_cmd_vel"/>
    <remap from="odom" to="/tb_control/wheel_odom"/>
    <remap from="scan" to="/scan"/>

    <param name="base_global_planner" type="string" value="navfn/NavfnROS" />
    <!--<param name="base_local_planner" value="base_local_planner/TrajectoryPlannerROS"/>-->

  </node>

</launch>
```

```yaml
shutdown_costmaps: false
controller_frequency: 3.0
controller_patience: 1.0
planner_frequency: 2.0
planner_patience: 1.0
oscillation_timeout: 10.0
oscillation_distance: 0.2
conservative_reset_dist: 0.10
```

```
NODES
  /
    amcl (amcl/amcl)
    map_server (map_server/map_server)
    move_base (move_base/move_base)

ROS_MASTER_URI=http://192.168.178.104:11311

process[map_server-1]: started with pid [18683]
process[amcl-2]: started with pid [18684]
process[move_base-3]: started with pid [18685]
[ WARN] [1645783315.590551135]: global_costmap: Pre-Hydro parameter "static_map" unused since "plugins" is provided
[ WARN] [1645783315.608581747]: global_costmap: Pre-Hydro parameter "map_type" unused since "plugins" is provided
[ INFO] [1645783315.620467072]: global_costmap: Using plugin "static_layer"
[ INFO] [1645783315.694057432]: Requesting the map...
[ INFO] [1645783316.042546840]: Resizing costmap to 2048 X 2048 at 0.050000 m/pix
[ INFO] [1645783316.133744966]: Received a 2048 X 2048 map at 0.050000 m/pix
[ INFO] [1645783316.181769679]: global_costmap: Using plugin "obstacle_layer"
[ INFO] [1645783316.275523501]:     Subscribed to Topics: laser_scan_sensor
[ INFO] [1645783316.588640010]: global_costmap: Using plugin "inflation_layer"
[ WARN] [1645783317.215623043]: local_costmap: Pre-Hydro parameter "map_type" unused since "plugins" is provided
[ INFO] [1645783317.227329875]: local_costmap: Using plugin "inflation_layer"
[ INFO] [1645783317.371615673]: local_costmap: Using plugin "obstacle_layer"
[ INFO] [1645783317.450979556]:     Subscribed to Topics:
[ INFO] [1645783318.004914636]: Created local_planner dwa_local_planner/DWAPlannerROS
[ INFO] [1645783318.045467769]: Sim period is set to 0.33
[ WARN] [1645783318.116815160]: Parameter max_rot_vel is deprecated (and will not load properly). Use max_vel_theta instead.
[ WARN] [1645783318.122244083]: Parameter min_rot_vel is deprecated (and will not load properly). Use min_vel_theta instead.
[ INFO] [1645783319.962087679]: Recovery behavior will clear layer 'obstacles'
[ INFO] [1645783320.055995326]: Recovery behavior will clear layer 'obstacles'
[ INFO] [1645783320.619905633]: odom received!
```

**Figure 4.11:** move_base_params file and move_base launch file with terminal window after launching the file



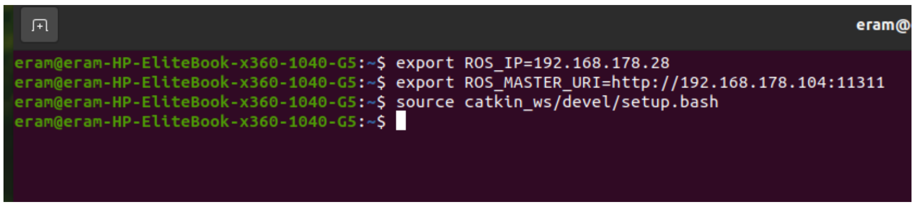
```cpp
#include <ros/ros.h>
#include <move_base_msgs/MoveBaseAction.h>
#include <actionlib/client/simple_action_client.h>

typedef actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction> MoveBaseClient;

int main(int argc, char** argv)
{
    ros::init(argc, argv, "navigation_goal");
    // Spin a thread
    MoveBaseClient ac("move_base", true);

    // Wait for the action server to come up
    ROS_INFO("Waiting for the move_base action server");
    ac.waitForServer(ros::Duration(5));

    ROS_INFO("Connected to move_base server");

    move_base_msgs::MoveBaseGoal goal;

    // Send goal pose
    goal.target_pose.header.frame_id = "map";
    goal.target_pose.header.stamp = ros::Time::now();

    // Do NOT modify the following for final submission.
    goal.target_pose.pose.position.x = -2.0;
    goal.target_pose.pose.position.y = 3.0;
    goal.target_pose.pose.position.z =  0.0;
    goal.target_pose.pose.orientation.x = 0.0;
    goal.target_pose.pose.orientation.y = 0.0;
    goal.target_pose.pose.orientation.z = 0.0;
    goal.target_pose.pose.orientation.w = 1.0;

    ROS_INFO("Sending goal");
    ac.sendGoal(goal);

    ac.waitForResult();

    if (ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED)
        ROS_INFO("Excellent! Your robot has reached the goal position.");
    else
        ROS_INFO("The robot failed to reach the goal position");

    return 0;
}
```

**Figure 4.12:** Simple action client node to send goals to the robot

# 5 Experimental Set up and Results

## 5.1 ROS master configuration

It is important to export information in order to run commands and use ROS tools like rviz on a local machine. On both the Ohmni's and the linux PC/Raspberry Pi that runs the rviz, we must set a variable "ROS MASTER URI" and "ROS IP" with the right Internet Protocol (IP) address in the bashrc file.



```
eram@eram-HP-EliteBook-x360-1040-G5:~$ export ROS_IP=192.168.178.28
eram@eram-HP-EliteBook-x360-1040-G5:~$ export ROS_MASTER_URI=http://192.168.178.104:11311
eram@eram-HP-EliteBook-x360-1040-G5:~$ source catkin_ws/devel/setup.bash
eram@eram-HP-EliteBook-x360-1040-G5:~$ 
```

**Figure 5.1:** Linux PC/RaspberryPi bashrc file

## 5.2 Test Case 1 - Mapping the room environment

The first test evaluates the accuracy of the mapping. The Ohmni Robot is used to map the environment, as previously mentioned. The robot has a LiDAR sensor attached on its tube link that allows it to sense its environment in two dimensions. The map is created using LiDAR data that is provided on the /scan topic. The Hector SLAM algorithm were made to follow a set path in a room, ensuring that the robot passes through a given area more than once and that data from LiDAR, was recorded. Because the LiDAR employed is a 2D LiDAR, scan points are generated in a horizontal plane at the height of the LiDAR. The LiDAR-based algorithms produces a two-dimensional map.

To start the mapping process, the ydlidar and Hector SLAM nodes are launched using the following command:

**roslaunch ydlidar all_nodes.launch**

**RViz** is a significant tool used in the mapping process. It will aid us in visualizing the map development process by allowing us to see what the robot is covering from the environment. Once the ydlidar starts spinning, rviz opens and it is imperative to set the fixed framed on rviz to /map if its not already set. The next step is to start the teleop_twist_keyboard.py script available in teleop_twist_keyboard to drive the robot around the lab/room environment using a wireless keyboard.

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

It is preferable to visit an area repeatedly and drive the robot slowly to obtain a high-quality map. During turns, it's also a good idea to halt the robot, spin it in the correct direction, and then resume driving. This allows the final map to be as similar to the actual design of the environment as possible. The map is formed after driving around the environment for a while, and it must be saved. To save the map, use the map_saver available on map_server package without terminating the Hector SLAM node.

```
rosrun map_server map_saver -f name_of_the_map
```

By specifying a filename, the map is saved with that name and a.pgm and.yaml file is created. The.pgm file contains the map image, whereas the.yaml file contains the map metadata. Now have a map of our environment according to the technique discussed. Portable Gray Map (PGM) which contains Occupancy Grid Map data. The Hector SLAM was not able to map the room accurately the first time. So of the obstacles were missed and also there was a drift in the map. Two maps of the same environment were created . The one on the left does catch all of the objects in the room, but only in part and with little detail as seen on the **Figure 5.2**. The room was remapped by moving the robot slowly, especially during turns, and closer to the objects, yielding a better map, as seen on **Figure 5.3**.

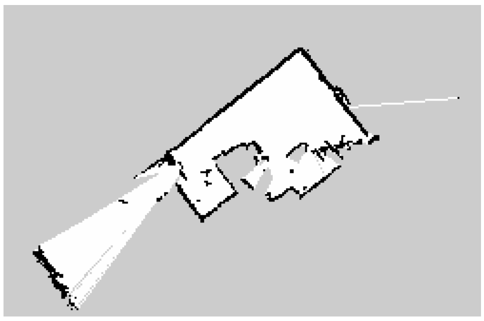


**Figure 5.2:** Test map of the room

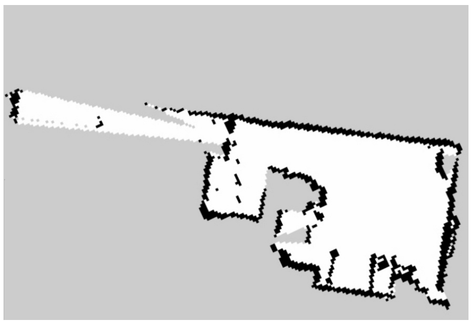


**Figure 5.3:** Final map of the room

## 5.3 Test Case 2 - Mapping the laboratory

The above mapping process was repeated in the laboratory environment and after many repetitions a fairly accurate map was produced. Here is how the laboratory looks like. It is rectangular in shape with workspaces of employees at small distances from each other. The figure below shows the final map and also one of the test maps generated during mapping the IAAS lab.

**Figure 5.4:** IAAS Laboratory

**Figure 5.5:** Laboratory view while entering

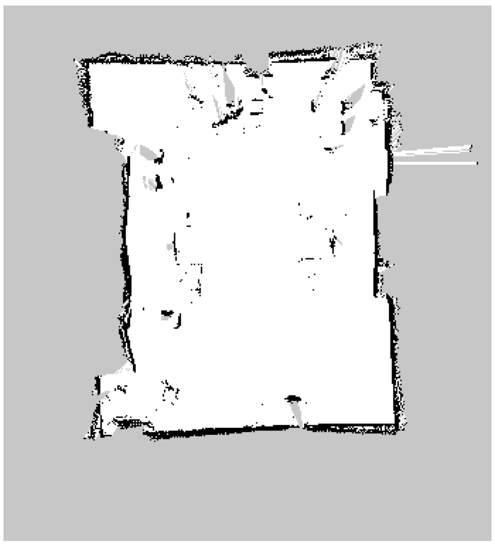


**Figure 5.6:** Test map of laboratory

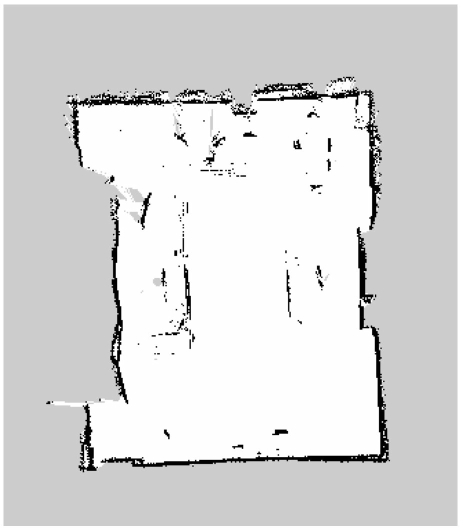


**Figure 5.7:** Final map of the laboratory

## 5.4 Localization of the robot on the resultant maps

To launch AMCL and start the localization process, we launch the startLocalization file

**roslaunch ohmni_2dnav startlocalization.launch**

We use the **Rviz 2D Pose Estimate** button to submit the robot's initial pose to the navigation system. Following that, the AMCL node will generate a large number of estimates (estimations of the robot's position and orientation in relation to the surroundings), as shown in the preceding picture. The final step is to begin moving the robot about the area using the teleop twist keyboard package until we observe the guesses begin to drop and become more concentrated. The figures below show that even when the robot's initial position is not quite accurate (off by up to 1 meter), the robot is able to localize itself within a few repetitions.

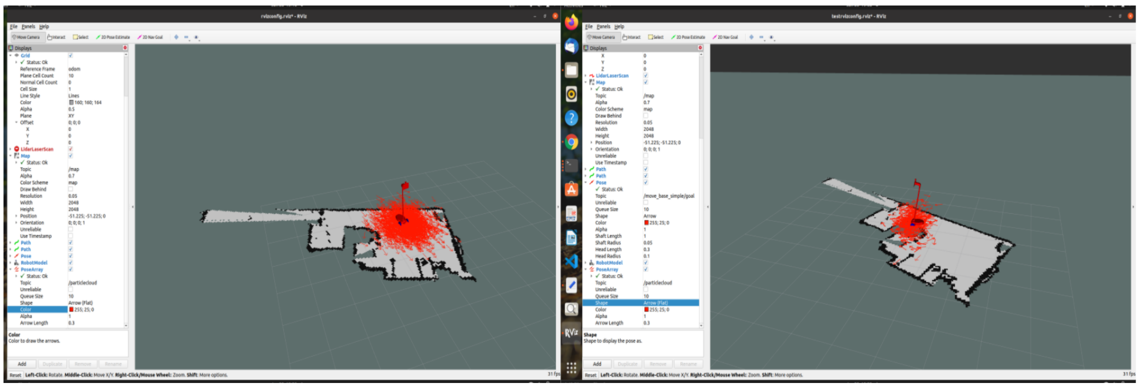


**Figure 5.8:** Localization of robot in room environment

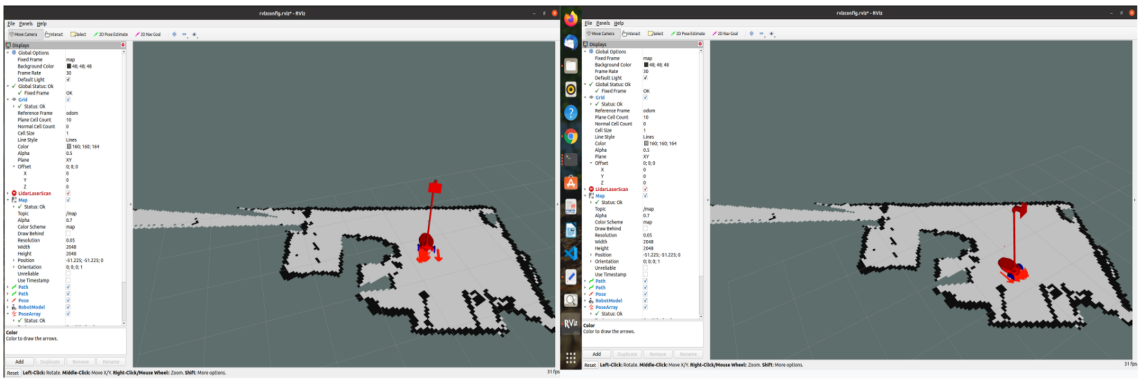


**Figure 5.9:** PoseArray particles reduces and robot is localized

The same process was repeated in the lab environment and the results are shown below :

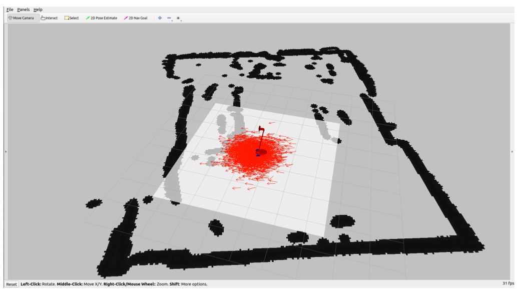


**Figure 5.10:** Robot visualized on both Global and Local costmaps

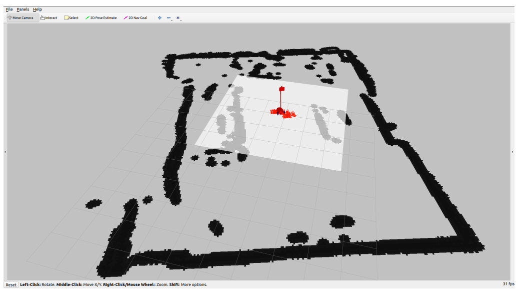
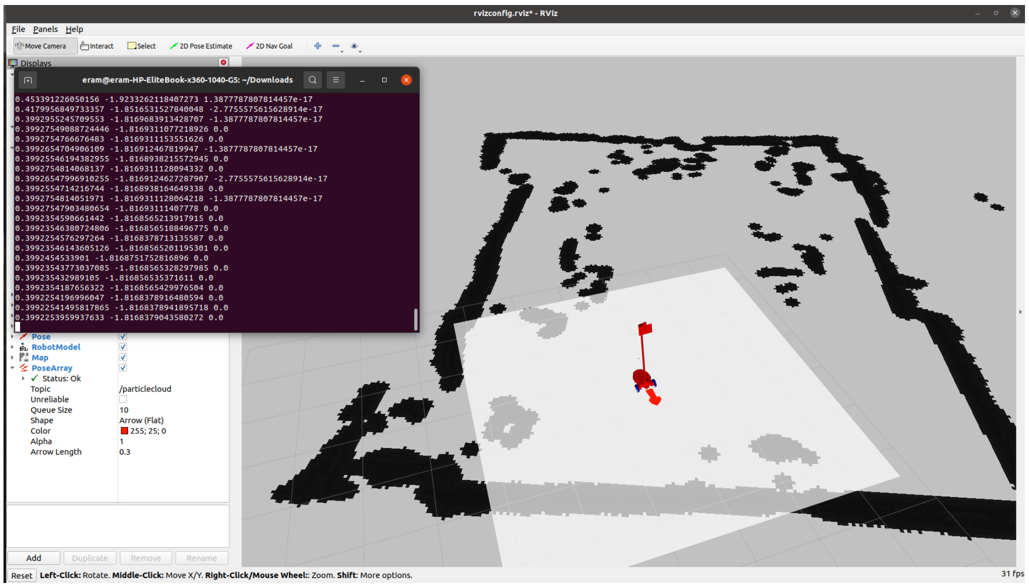


**Figure 5.11:** After moving the robot for a while in the lab


rvizconfig.rviz* - RViz
File
Panels
Help
Move Camera
Interact
Select
2D Pose Estimate
2D Nav Goal
Displays
eram@eram-HP-EliteBook-x360-1040-G5: ~/Downloads
Add
Duplicate
Remove
Rename
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.
31 fps

```
0.453391226050156 -1.9233262118407273 1.3877787807814457e-17
0.4179956849733357 -1.8516531527840048 -2.7755575615628914e-17
0.3992955245709553 -1.8169683913428707 -1.3877787807814457e-17
0.39927549088724446 -1.8169311077218926 0.0
0.3992754766676483 -1.8169311153551626 0.0
0.3992654704906109 -1.816912467819947 -1.3877787807814457e-17
0.39925546194382955 -1.8168938215572945 0.0
0.3992754814068137 -1.8169311128094332 0.0
0.39926547996910255 -1.8169124627287907 -2.7755575615628914e-17
0.3992554714216744 -1.8168938164649338 0.0
0.3992754814051971 -1.8169311128064218 -1.3877787807814457e-17
0.39927547903480654 -1.81693111407778 0.0
0.3992354590661442 -1.8168565213917915 0.0
0.39923546380724806 -1.8168565188496775 0.0
0.3992254576297264 -1.8168378713135587 0.0
0.39923546143605126 -1.8168565201195301 0.0
0.3992454533901 -1.8168751752816896 0.0
0.39923543773037085 -1.8168565328297985 0.0
0.399235432989105 -1.816856535371611 0.0
0.3992354187656322 -1.8168565429976504 0.0
0.3992254196996047 -1.8168378916480594 0.0
0.39922541495817865 -1.8168378941895718 0.0
0.3992253959937633 -1.8168379043580272 0.0
```

| | |
|---|---|
| Pose | ✓ |
| RobotModel | ✓ |
| Map | ✓ |
| PoseArray | ✓ |
| ✓ Status: Ok | |
| Topic | /particlecloud |
| Unreliable | ☐ |
| Queue Size | 10 |
| Shape | Arrow (Flat) |
| Color | 🟥 255; 25; 0 |
| Alpha | 1 |
| Arrow Length | 0.3 |

**Figure 5.12:** Robot localized and PoseArray particles reduced to a large single arrow

## 5.5 Map based autonomous navigation

To start the map based autonomous navigation first launch the configuration and the move base file using the following commands :

**roslaunch ohmni_2dnav ohmni_configuration.launch**

Next,

**roslaunch ohmni_2dnav move_base.launch**

To start the navigation to goal node, run the following command:

**rosrun ohmni_2dnav navigation_goal**

The robot would move towards the goal and then once it reaches the goal the success message will be displayed on the terminal window. The goal can also be given on Rviz as shown in the **figures 5.13 and 5.14**.

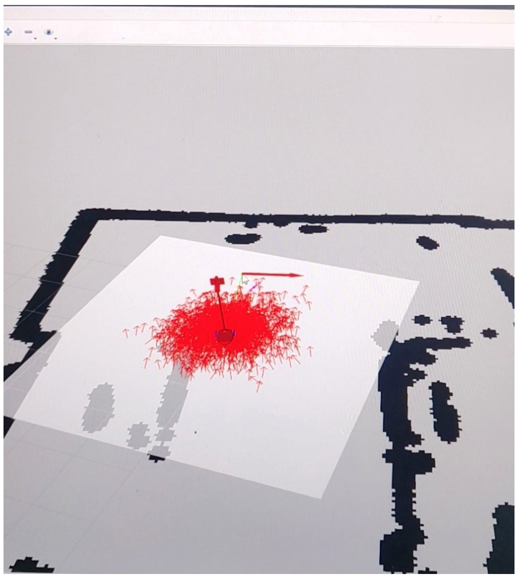
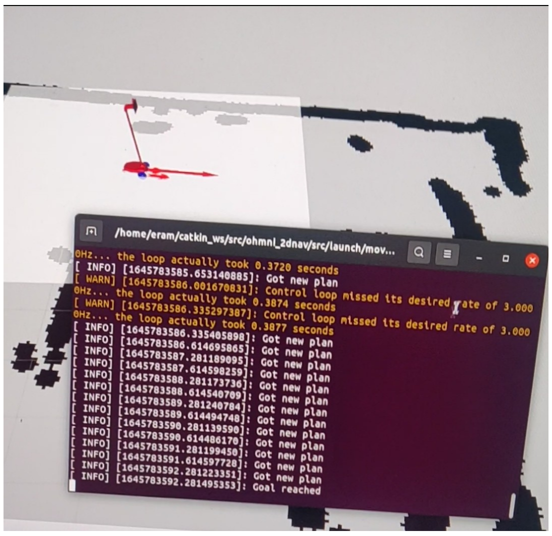


**Figure 5.13:** Giving goal position on Rviz



```
/home/eram/catkin_ws/src/ohmni_2dnav/src/launch/mov...
0Hz... the loop actually took 0.3720 seconds
[ INFO] [1645783585.653140885]: Got new plan
[ WARN] [1645783586.001670831]: Control loop missed its desired rate of 3.000
0Hz... the loop actually took 0.3874 seconds
[ WARN] [1645783586.335297387]: Control loop missed its desired rate of 3.000
0Hz... the loop actually took 0.3877 seconds
[ INFO] [1645783586.335405898]: Got new plan
[ INFO] [1645783586.614695865]: Got new plan
[ INFO] [1645783587.281189095]: Got new plan
[ INFO] [1645783587.614598259]: Got new plan
[ INFO] [1645783588.281173736]: Got new plan
[ INFO] [1645783588.614540709]: Got new plan
[ INFO] [1645783589.281240784]: Got new plan
[ INFO] [1645783589.614494748]: Got new plan
[ INFO] [1645783590.281139590]: Got new plan
[ INFO] [1645783590.614486170]: Got new plan
[ INFO] [1645783591.281199450]: Got new plan
[ INFO] [1645783591.614597728]: Got new plan
[ INFO] [1645783592.281223351]: Got new plan
[ INFO] [1645783592.281495353]: Goal reached
```

**Figure 5.14:** Goal Reached

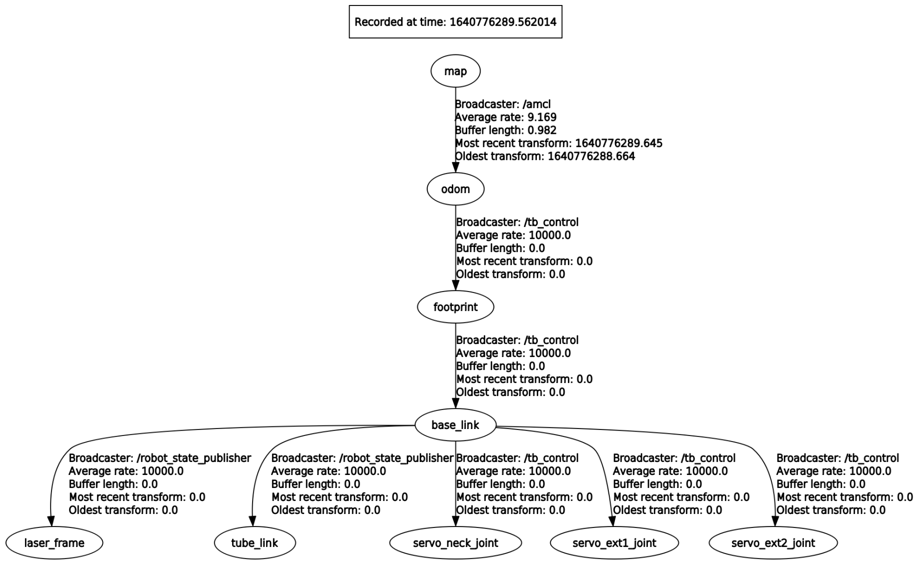


**Figure 5.15:** ROS frame generated of the robot model

## 5.6 Performance Analysis

#### 5.6.0.1 System Performance

Once the navigation stack is running and odometry data is received, it is expected that the robot starts navigating in the environment provided to it by the map. But that is not always the case. There was one such error which occurred during the project which is worth mentioning.

**"Costmap2DROS transform timeout. Current time: 164253455.8331, global_pose stamp: 16453523552.6151, tolerance: 0.2000"**

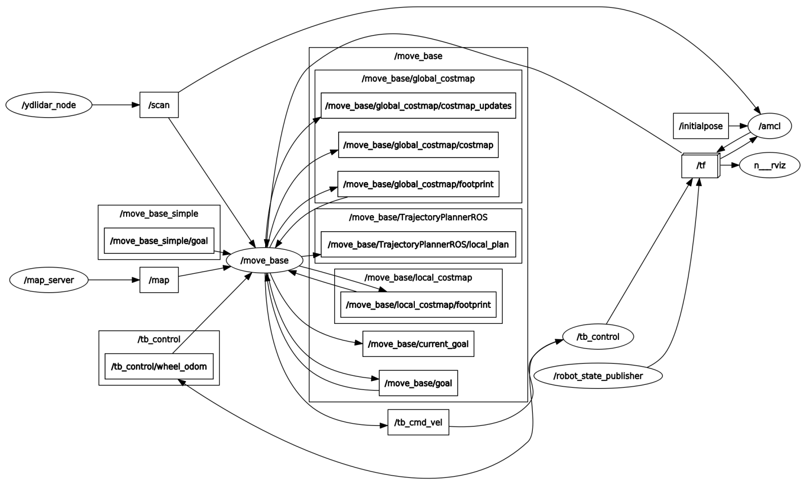
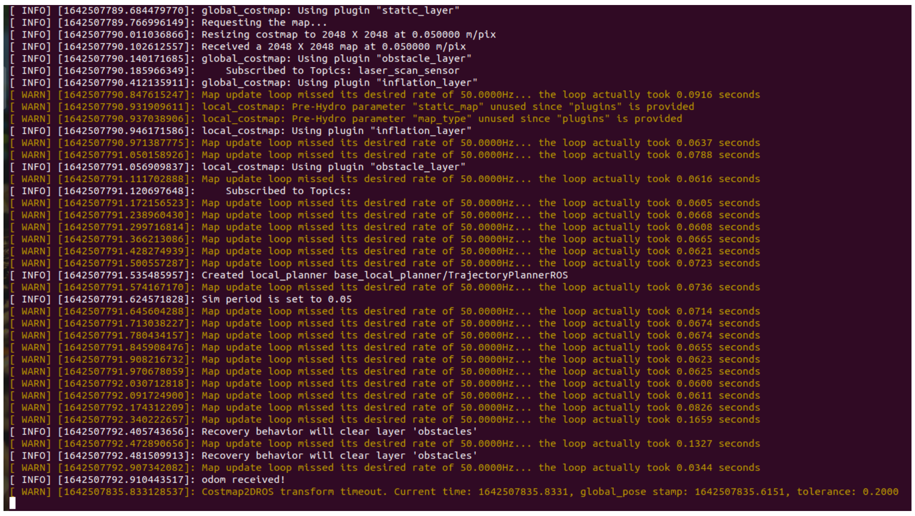


**Figure 5.16:** ROS graph with the all the nodes while the move_base, amcl and robot is navigating

Even when there was no problem with the stack and all the necessary data was received correctly, the robot was exhibiting recovery behavior and could not navigate to the goal. Several changes were carried out in the configuration files, like changing the obstacle and raytrace range in the common_costmap.params file, setting the update and publish frequency to low values. So that the pressure on the system is reduced, the transform tolerance was increased to a higher value. There is enough time available for the transforms to take place by doing so. Yet the error kept coming back. Ultimately, the solution provided more RAM while running the navigation stack and Rviz. So the error is likely to occur again if enough RAM is not available, especially when trying to run the code on RaspberryPi. One should also close all other applications running on the system before performing the autonomous navigation; otherwise, a rotate recovery behavior of the robot will start. It was a finding while performing testing—the error is shown in figure 5.17.



```
[ INFO] [1642507789.684479770]: global_costmap: Using plugin "static_layer"
[ INFO] [1642507789.766996149]: Requesting the map...
[ INFO] [1642507790.011036866]: Resizing costmap to 2048 X 2048 at 0.050000 m/pix
[ INFO] [1642507790.102612557]: Received a 2048 X 2048 map at 0.050000 m/pix
[ INFO] [1642507790.140171685]: global_costmap: Using plugin "obstacle_layer"
[ INFO] [1642507790.185966349]:     Subscribed to Topics: laser_scan_sensor
[ INFO] [1642507790.412135911]: global_costmap: Using plugin "inflation_layer"
[ WARN] [1642507790.847615247]: Map update loop missed its desired rate of 50.0000Hz... the loop actually took 0.0916 seconds
[ WARN] [1642507790.931909611]: local_costmap: Pre-Hydro parameter "static_map" unused since "plugins" is provided
[ WARN] [1642507790.937038906]: local_costmap: Pre-Hydro parameter "map_type" unused since "plugins" is provided
[ INFO] [1642507790.946171586]: local_costmap: Using plugin "inflation_layer"
[ WARN] [1642507790.971387775]: Map update loop missed its desired rate of 50.0000Hz... the loop actually took 0.0637 seconds
[ WARN] [1642507791.050158926]: Map update loop missed its desired rate of 50.0000Hz... the loop actually took 0.0788 seconds
[ INFO] [1642507791.056909837]: local_costmap: Using plugin "obstacle_layer"
[ WARN] [1642507791.111702888]: Map update loop missed its desired rate of 50.0000Hz... the loop actually took 0.0616 seconds
[ INFO] [1642507791.120697648]:     Subscribed to Topics:
[ WARN] [1642507791.172156523]: Map update loop missed its desired rate of 50.0000Hz... the loop actually took 0.0605 seconds
[ WARN] [1642507791.238960430]: Map update loop missed its desired rate of 50.0000Hz... the loop actually took 0.0668 seconds
[ WARN] [1642507791.299716814]: Map update loop missed its desired rate of 50.0000Hz... the loop actually took 0.0608 seconds
[ WARN] [1642507791.366213086]: Map update loop missed its desired rate of 50.0000Hz... the loop actually took 0.0665 seconds
[ WARN] [1642507791.428274939]: Map update loop missed its desired rate of 50.0000Hz... the loop actually took 0.0621 seconds
[ WARN] [1642507791.500557287]: Map update loop missed its desired rate of 50.0000Hz... the loop actually took 0.0723 seconds
[ INFO] [1642507791.535485957]: Created local_planner base_local_planner/TrajectoryPlannerROS
[ WARN] [1642507791.574167170]: Map update loop missed its desired rate of 50.0000Hz... the loop actually took 0.0736 seconds
[ INFO] [1642507791.624571828]: Sim period is set to 0.05
[ WARN] [1642507791.645604288]: Map update loop missed its desired rate of 50.0000Hz... the loop actually took 0.0714 seconds
[ WARN] [1642507791.713038227]: Map update loop missed its desired rate of 50.0000Hz... the loop actually took 0.0674 seconds
[ WARN] [1642507791.780434157]: Map update loop missed its desired rate of 50.0000Hz... the loop actually took 0.0674 seconds
[ WARN] [1642507791.845908476]: Map update loop missed its desired rate of 50.0000Hz... the loop actually took 0.0655 seconds
[ WARN] [1642507791.908216732]: Map update loop missed its desired rate of 50.0000Hz... the loop actually took 0.0623 seconds
[ WARN] [1642507791.970678059]: Map update loop missed its desired rate of 50.0000Hz... the loop actually took 0.0625 seconds
[ WARN] [1642507792.030712818]: Map update loop missed its desired rate of 50.0000Hz... the loop actually took 0.0600 seconds
[ WARN] [1642507792.091724900]: Map update loop missed its desired rate of 50.0000Hz... the loop actually took 0.0611 seconds
[ WARN] [1642507792.174312209]: Map update loop missed its desired rate of 50.0000Hz... the loop actually took 0.0826 seconds
[ WARN] [1642507792.340222657]: Map update loop missed its desired rate of 50.0000Hz... the loop actually took 0.1659 seconds
[ INFO] [1642507792.405743656]: Recovery behavior will clear layer 'obstacles'
[ WARN] [1642507792.472890656]: Map update loop missed its desired rate of 50.0000Hz... the loop actually took 0.1327 seconds
[ INFO] [1642507792.481509913]: Recovery behavior will clear layer 'obstacles'
[ WARN] [1642507792.907342082]: Map update loop missed its desired rate of 50.0000Hz... the loop actually took 0.0344 seconds
[ INFO] [1642507792.910443517]: odom received!
[ WARN] [1642507835.833128537]: Costmap2DROS transform timeout. Current time: 1642507835.8331, global_pose stamp: 1642507835.6151, tolerance: 0.2000
```

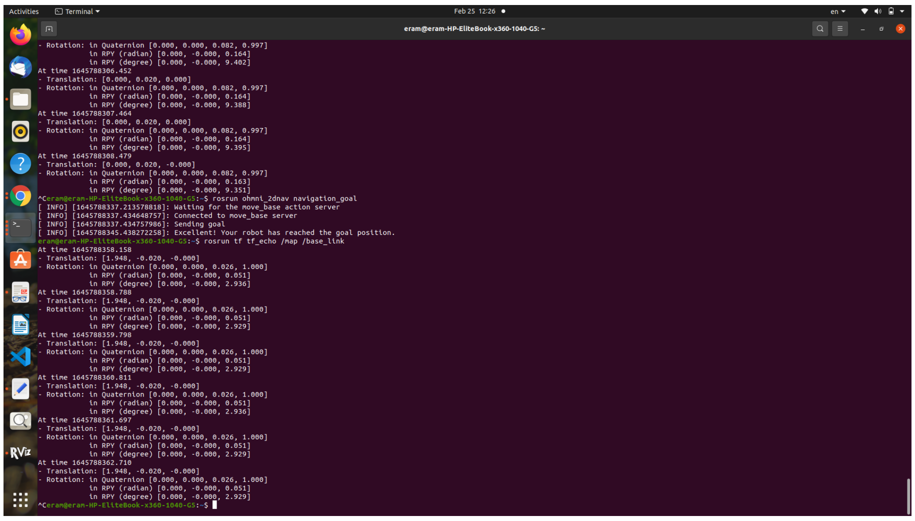
**Figure 5.17:** Costmap2DROS transform error due to low level RAM

#### 5.6.0.2 Testing map-based navigation behaviour at different velocities

The navigation setup and code was tested by giving the robot different goal coordinates at different speeds. The maximum speed obtainable by the robot is increased slightly in each iteration and the coordinates of initial point and final position of the robot is captured. The following steps are carried out in this test.

- In the first iteration, the max_vel parameter is changed in the dwa_local_planner config file to 0.1 m/secs.
- Then, the goal coordinates were updated in the navigation_goal.cpp file.
- The setup was then executed using the instructions given in **Section 5.5**.
- The next step was to capture the initial position of the robot on the map by running the following command.

  **rosrun tf tf_echo /map /base_link**
- Then the navigation_goal file was ran.
- Once the robot reached it goal and the success message displayed on the terminal, the position of the robot on the map is captured again.
- The above steps were repeated for different velocity values and the difference between the desired distance travelled by the robot and the actual distance is calculated in each case. This gives the error for each goal point.
- The various initial coordinates, goal coordinates and calculated distances are listed in the **Tables 5.18 and 5.19**.
- The figure below shows the initial and actual goal coordinates of the robot on the map when the test was run for max_vel 0.5 m/s and desired goal coordinate of (2,0).



```
- Rotation: in Quaternion [0.000, 0.000, 0.082, 0.997]
            in RPY (radian) [0.000, -0.000, 0.164]
            in RPY (degree) [0.000, -0.000, 9.402]
At time 1645788306.452
- Translation: [0.000, 0.020, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.082, 0.997]
            in RPY (radian) [0.000, -0.000, 0.164]
            in RPY (degree) [0.000, -0.000, 9.388]
At time 1645788307.464
- Translation: [0.000, 0.020, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.082, 0.997]
            in RPY (radian) [0.000, -0.000, 0.164]
            in RPY (degree) [0.000, -0.000, 9.395]
At time 1645788308.479
- Translation: [0.000, 0.020, -0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.082, 0.997]
            in RPY (radian) [0.000, -0.000, 0.163]
            in RPY (degree) [0.000, -0.000, 9.351]
^Ceram@eram-HP-EliteBook-x360-1040-G5:~$ rosrun ohmni_2dnav navigation_goal
[ INFO] [1645788337.213578818]: Waiting for the move_base action server
[ INFO] [1645788337.434648757]: Connected to move_base server
[ INFO] [1645788337.434757986]: Sending goal
[ INFO] [1645788345.438272258]: Excellent! Your robot has reached the goal position.
eram@eram-HP-EliteBook-x360-1040-G5:~$ rosrun tf tf_echo /map /base_link
At time 1645788358.158
- Translation: [1.948, -0.020, -0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.026, 1.000]
            in RPY (radian) [0.000, -0.000, 0.051]
            in RPY (degree) [0.000, -0.000, 2.936]
At time 1645788358.788
- Translation: [1.948, -0.020, -0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.026, 1.000]
            in RPY (radian) [0.000, -0.000, 0.051]
            in RPY (degree) [0.000, -0.000, 2.929]
At time 1645788359.798
- Translation: [1.948, -0.020, -0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.026, 1.000]
            in RPY (radian) [0.000, -0.000, 0.051]
            in RPY (degree) [0.000, -0.000, 2.929]
At time 1645788360.811
- Translation: [1.948, -0.020, -0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.026, 1.000]
            in RPY (radian) [0.000, -0.000, 0.051]
            in RPY (degree) [0.000, -0.000, 2.936]
At time 1645788361.697
- Translation: [1.948, -0.020, -0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.026, 1.000]
            in RPY (radian) [0.000, -0.000, 0.051]
            in RPY (degree) [0.000, -0.000, 2.929]
At time 1645788362.710
- Translation: [1.948, -0.020, -0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.026, 1.000]
            in RPY (radian) [0.000, -0.000, 0.051]
            in RPY (degree) [0.000, -0.000, 2.929]
^Ceram@eram-HP-EliteBook-x360-1040-G5:~$
```

**Figure 5.18:** Initial and actual goal coordinates of the robot on the map when the test was run for max_vel 0.5 m/s and desired goal coordinate of (2,0)

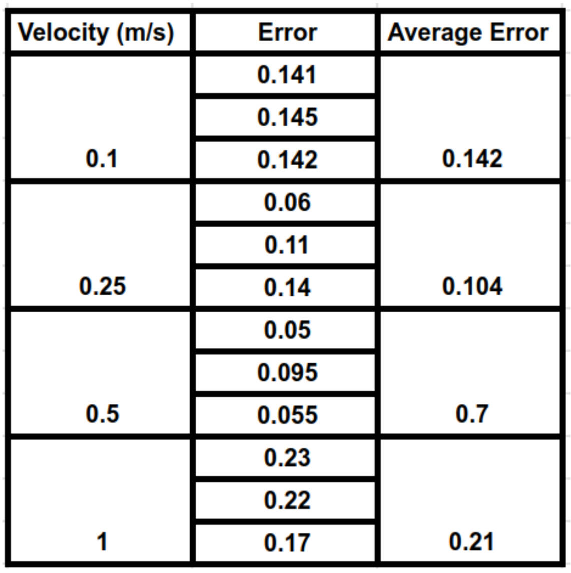
| Velocity (m/s) | Initial Position (X,Y) | Desired Goal (X,Y) | Desired Distance (D) |
|---|---|---|---|
| | (0,0.020) | (2,0) | 2.001 |
| | (0.019,0.013) | (-2.329,-3.029) | 3.842 |
| 0.1 | (-0.023,0.033) | (-4.098,2.017) | 4.5323 |
| | (-0.024,0.013) | (2,0) | 2.024 |
| | (-0.052,0.031) | (-2.579,-3.423) | 4.28 |
| 0.25 | (0.025,-0.071) | (-4.127,2.021) | 4.649 |
| | (0,0.020) | (2,0) | 2.001 |
| | (0.015,0.025) | (-2.623,-3.113) | 4.099 |
| 0.5 | (0.021,0.093) | (-3.711,3.743) | 5.589 |
| | (0.011,0.013) | (2,3.029) | 3.613 |
| | (-0.020,0.033) | (-2.329,-3.029) | 3.835 |
| 1 | (0,0.020) | (-4.098,2.017) | 4.44 |

**Table 5.1:** Desired goal coordinates and corresponding desired distances (meters)

| Velocity (m/s) | Initial Position (X,Y) | Actual Goal (X,Y) | Actual Distance (D) |
|---|---|---|---|
| | (0,0.020) | (1.857,0.059) | 1.859 |
| | (0.019,0.013) | (-2.022,-3.070) | 3.697 |
| 0.1 | (-0.023,0.033) | (-3.988,1.917) | 4.39 |
| | (-0.024,0.013) | (1.938,-0.045) | 1.963 |
| | (-0.052,0.031) | (-2.550,-3.313) | 4.174 |
| 0.25 | (0.025,-0.071) | (-4.027,1.909) | 4.509 |
| | (0,0.020) | (1.948,-0.020) | 1.948 |
| | (0.015,0.025) | (-2.303,-3.240) | 4.004 |
| 0.5 | (0.021,0.093) | (-3.932,3.965) | 5.533 |
| | (0.011,0.013) | (2.298,3.102) | 3.843 |
| | (-0.020,0.033) | (-2.275,-3.338) | 4.056 |
| 1 | (0,0.020) | (-4.201,2.198) | 4.613 |

**Table 5.2:** Actual goal coordinates and corresponding actual distances (meters)

The computed errors are listed on the **Table 5.3**. It was observed that when the robot was given a velocity of 0.1 m/secs, it missed the goal by a few centimeters and also it took a lot of time to barely reach the desired point. As the speed was increased to 0.25 m/s and then 0.5 m/s there was a considerable decrease in the error and it was noted that by setting the maximum speed to 0.5 m/s the error was minimum. After that when there was an increase in the speed of the robot to 1.0 m/s, the robot tend to cross the goal point and maximum error occurred.



| Velocity (m/s) | Error | Average Error |
|---|---|---|
| | 0.141 | |
| | 0.145 | |
| 0.1 | 0.142 | 0.142 |
| | 0.06 | |
| | 0.11 | |
| 0.25 | 0.14 | 0.104 |
| | 0.05 | |
| | 0.095 | |
| 0.5 | 0.055 | 0.7 |
| | 0.23 | |
| | 0.22 | |
| 1 | 0.17 | 0.21 |

**Table 5.3:** Error corresponding to different velocities of the robot

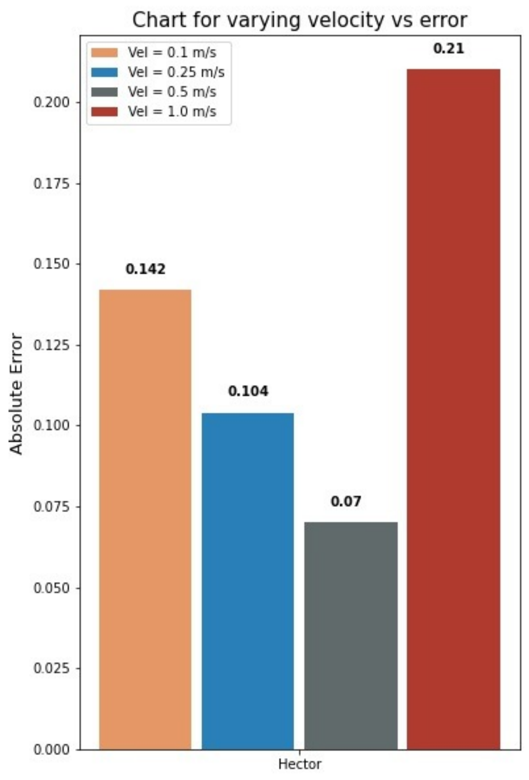


**Figure 5.19:** Chart showing errors corresponding to different velocities of the robot

# 6 Conclusion

In this thesis, LiDAR-based SLAM is implemented on a telepresence robot, which is then capable of autonomous navigation using the ROS navigation stack. The robot's whole navigation is based on a recorded map built by Hector SLAM. With the help of a laser sensor and a map, the robot can travel across the lab environment. It is capable of successfully navigating through the map and around any charted obstacles. The map is a 2D grid-based SLAM map, derived from LiDAR sensor data and an approximation of the robot's position. The robot's full navigation system is divided into two different planners.

The global planner determines the quickest path from an estimated starting point to a specified goal. The computation is based on all of the obstacles that have been charted inside the recorded map and the Dijkstra algorithm. The second planner is the local planner, whose job it is to find the optimal path based on the current events. If the LiDAR sensor detects an unknown obstacle, it will redirect the local route. It will also re-plan the course if following the path without ranking is not achievable. The velocity commands are also calculated by the local planner. A Monte Carlo localization approach is employed to locate the robot within the map and throughout the trip. The algorithm scans the environment to see if the measurements match the documented obstacles on the map. The robot's motion model is used to match the scanned measurements of obstacles with the one's present on the map. Based on these data, the program can estimate its current position and direction on the map.

## Findings

As seen from the first test - mapping the room and lab environment, for the pure LiDAR-based SLAM algorithm, Hector SLAM can generate a 2D map of the lab and room with accurate wall boundaries. Hector SLAM can learn the full features of the environment, and it relies entirely on observed characteristics when there are no new features. (e.g., the long passage right in the middle of the lab with identical tables placed on both sides) as shown in **Fig. 5.4**, the algorithm fails to update the map even though the robot moves through the environment. There was one more problem with the accuracy of the map where it has a gap in the far end corners, as the LiDAR doesn't detect glass windows behind the divider/shoji screen efficiently. Hence, results in loss of map data.

Algorithms for indoor robots used in homes and offices for assistance and organizing are chosen based on the processor's needed operation speed and available processing capability. An autonomous robot developed for assisting people in a laboratory must be usable at various speeds based on the employees' needs and preferences. Because SLAM is an optimization of the perception section of the autonomous pipeline, as stated in Chapter 2, it should be able to produce a map with as many learned characteristics (obstacles in some situations) while simultaneously predicting the robot's position. Given the above reasons, the autonomous navigation solution provided in this project was tested at various operating speeds of the robot. **Fig 5.19** shows the errors in the estimated distance (meters) traveled by the robot at a given speed. It was observed that at very low and comparatively higher velocities (> 0.5 m/s), the computed error increases. Therefore the optimal speed was chosen to be 0.5 m/s.

## Major Challenges

There were a few challenges faced throughout the course of the Master Thesis that are listed below:

- Due to the ongoing corona situation, access to the laboratory was denied, and the robot's work was tested in a room environment. The area of the room was small due to which the navigation stack specifications had to be tuned to a great extent, and all the parameters like obstacle range and distance to be covered by the robot (giving nearby goal coordinates) were set to very small values to keep up with the limited space. Even then, the robot would collide with the obstacles or the wall as there are too close and packed together. When the setup was shifted to a larger surrounding of the laboratory, the performance was improved.

- Hector SLAM believes the robot is not moving (even if it is) when the laser scan readings at the "long corridor" are nearly identical. In this case, the large clear pathway in the middle of the lab between the work-spaces of the employees behaves like a **long corridor**. This problem is also explicitly stated in ROS answers, and because hector SLAM is not using odometry (except roll and pitch), it cannot be fixed.

- Unfortunately, the laser does not reflect from glass, so performance was poor in a building with glass doors and walls. Reflections are typically reproducible, and to the LiDAR/SLAM system, it appears that there is "a room behind the mirror," with the "room" being used for scan matching. Of course, if a robot tries to drive into this room, things become a bit unpleasant (or if phantom obstacles are generated in otherwise free space). Actually, recognizing reflective/mirror surfaces can be difficult, and it is not known that any open-source ROS solution is available for that purpose. One possible solution is to use another sensor with a different sensing modality, for example, an ultrasonic sensor. This is obviously beneficial, but integrating that data with the rest of the software (SLAM as well as path planning) can be a difficult undertaking. Furthermore, when the angle of incidence is at a particular degree, the laser beam sometimes reflects from the glass. The laser does reflect back at 0 degrees of incidence, and the phenomenon is known as total internal reflection. This phenomenon causes the LiDAR to produce varied readings at different degrees of incidence, resulting in an inconsistent map because it further confuses the mapping algorithms.

  Having said this, a region in the laboratory (wall behind the "shoji" screen/room divider in case of the lab) has glass walls, and after repeated iterations of mapping, it was not properly detected and mapped.

- The "angle too large" issue: Hector SLAM gives the "angle too large" warning when the robot turns too many times, in a short period of time. However, the warning itself is inconsistent; some tests were run to see how large an angle/turn would result in this error, but there was no definite pattern. However, the map would drift every time after this warning. "Map missed the desired loop:" warning occurs while running the navigation stack, and there is no particular reason for it which could be determined. All the parameters related to loop time have been adjusted suited to the environment. The warning does not affect the performance and is not persistent. So it is safe to mention that the warning can be ignored.

## Future Work

- For the return of the robot to its original position or the location of the docking station the robot's localization in the surroundings must be near perfect for this. The robot's position on the map will change as it navigates the region surrounding its goal location. To return it to its original place or the docking station, we believe the beginning position should be reset on the map; otherwise, the localization errors would accumulate, and the robot will locate itself badly, frequently missing its intended location. So far, RViz has been used to set the initial position.
- Multiple destinations can be sent to the robot as checkpoints before it reaches its final goal position. It is possible to make it follow a pre-determined trajectory. A multitude of things can be placed between the robot and its target, and it is feasible to demonstrate that the robot can navigate past them to reach its destination. One can change the position of the items on their return path to observe how the robot performs in attempting to return to its previous position.
- In an effort to make the telepresence robot autonomous using its ability to perform SLAM, one can position an item on a tray on the robot and provide it with a set of coordinates to deliver the item to an individual at that position, which represents Human-Robot Collaboration (HRC). Also, one can provide a time delay so that the item can be retrieved by the individual. The robot can then navigate back to its previous position.

# Bibliography

[AZ18] O. Agunbiade, T. Zuva. “A review: simultaneous localization and mapping in application to autonomous robot”. In: (2018) (cit. on p. 31).

[FD06] J. Forlizzi, C. DiSalvo. “Service robots in the domestic environment: a study of the roomba vacuum in the home”. In: *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*. 2006, pp. 258–265 (cit. on p. 17).

[FOL76] F. FOLE. “From da Vinci to the present-A review of airscrew theory for helicopters, propellers, windmills, and engines”. In: *9th Fluid and PlasmaDynamics Conference*. 1976, p. 367 (cit. on p. 17).

[GKSB10] G. Grisetti, R. Kümmerle, C. Stachniss, W. Burgard. “A tutorial on graph-based SLAM”. In: *IEEE Intelligent Transportation Systems Magazine* 2.4 (2010), pp. 31–43 (cit. on p. 20).

[HM11] F. Hashikawa, K. Morioka. “Mobile robot navigation based on interactive SLAM with an intelligent space”. In: *2011 8th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. 2011, pp. 788–789. DOI: 10.1109/URAI.2011.6146017 (cit. on p. 32).

[HT12] A. Hodges, A. Turing. *The Enigma: The Enigma*. 2012 (cit. on p. 17).

[JBM+19] A. Juneja, L. Bhandari, H. Mohammadbagherpoor, A. Singh, E. Grant. “A Comparative Study of SLAM Algorithms for Indoor Navigation of Autonomous Wheelchairs”. In: *2019 IEEE International Conference on Cyborg and Bionic Systems (CBS)*. 2019, pp. 261–266. DOI: 10.1109/CBS46900.2019.9114512 (cit. on p. 32).

[KUTY00] S. Kondoh, Y. Umeda, T. Tomiyama, H. Yoshikawa. “Self organization of cellular manufacturing systems”. In: *CIRP Annals* 49.1 (2000), pp. 347–350 (cit. on p. 17).

[labna] ohmni labs. *ohmni developer*. na (cit. on p. 26).

[LH02] J.-H. Lee, H. Hashimoto. “Intelligent space—concept and contents”. In: *Advanced Robotics* 16.3 (2002), pp. 265–280 (cit. on p. 31).

[LS18] Y. Li, C. Shi. “Localization and Navigation for Indoor Mobile Robot Based on ROS”. In: *2018 Chinese Automation Congress (CAC)*. 2018, pp. 1135–1139. DOI: 10.1109/CAC.2018.8623225 (cit. on p. 32).

[LT10] J. Levinson, S. Thrun. “Robust vehicle localization in urban environments using probabilistic maps”. In: *2010 IEEE international conference on robotics and automation*. IEEE. 2010, pp. 4372–4378 (cit. on p. 19).

[MRM20] R. K. Megalingam, A. Rajendraprasad, S. K. Manoharan. “Comparison of Planned Path and Travelled Path Using ROS Navigation Stack”. In: *2020 International Conference for Emerging Technology (INCET)*. 2020, pp. 1–6. DOI: 10.1109/INCET49848.2020.9154132 (cit. on p. 32).

[poi17] tutorials point. *docker tutorial*. 2017 (cit. on p. 26).

[QCG+09] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, et al. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5 (cit. on p. 31).

[QGS15] M. Quigley, B. Gerkey, W. D. Smart. *Programming Robots with ROS: a practical introduction to the Robot Operating System*. Ö'Reilly Media, Inc.", 2015 (cit. on pp. 21, 31).

[RB03] S. Riisgaard, M. R. Blas. "SLAM for Dummies". In: *A Tutorial Approach to Simultaneous Localization and Mapping* 22.1-127 (2003), p. 126 (cit. on p. 27).

[RMV19] W. P. N. dos Reis, O. Morandin, K. C. T. Vivaldini. "A Quantitative Study of Tuning ROS Adaptive Monte Carlo Localization Parameters and their Effect on an AGV Localization". In: *2019 19th International Conference on Advanced Robotics (ICAR)*. 2019, pp. 302–307. DOI: 10.1109/ICAR46387.2019.8981601 (cit. on p. 31).

[ROB16] I. L. ROBOTICS. "A DPDHL perspective on implications and use cases for the logistics industry". In: *DHL March* (2016) (cit. on p. 18).

[SATS20] S. Saat, W. Abd Rashid, M. Tumari, M. Saealal. "Hectorslam 2d Mapping for Simultaneous Localization and Mapping (slam)". In: *Journal of Physics: Conference Series*. Vol. 1529. 4. IOP Publishing. 2020, p. 042032 (cit. on p. 31).

[soulea] B. l. open source. *Human–robot interaction - Wikipedia*. not available (cit. on p. 12).

[souleb] B. l. open source. *OhmniRobot*. not available (cit. on p. 13).

[soulec] B. l. open source. *open source documentation for ros*. not available (cit. on p. 22).

[souled] B. l. open source. *open source documentation for ros*. not available (cit. on p. 22).

[soulee] B. l. open source. *open source documentation for ros*. not available (cit. on pp. 23, 31).

[soulef] B. l. open source. *open source documentation for ros*. not available (cit. on p. 23).

[TMD+06] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, et al. "Stanley: The robot that won the DARPA Grand Challenge". In: *Journal of field Robotics* 23.9 (2006), pp. 661–692 (cit. on p. 18).

[Urm+08] C. Urmson et al. "Self-driving cars and the urban challenge". In: *IEEE Intelligent Systems* 23.2 (2008), pp. 66–68 (cit. on p. 17).

[XGG+19] Z. Xuexi, L. Guokun, F. Genping, X. Dongliang, L. Shiliu. "SLAM Algorithm Analysis of Mobile Robot Based on Lidar". In: *2019 Chinese Control Conference (CCC)*. 2019, pp. 4739–4745. DOI: 10.23919/ChiCC.2019.8866200 (cit. on p. 32).

[XZZH15] Q. Xu, J. Zhao, C. Zhang, F. He. "Design and implementation of an ROS based autonomous navigation system". In: *2015 IEEE International Conference on Mechatronics and Automation (ICMA)*. 2015, pp. 2220–2225. DOI: 10.1109/ICMA.2015.7237831 (cit. on p. 32).

[YLC+19] J. Yang, Y. Li, L. Cao, Y. Jiang, L. Sun, Q. Xie. "A survey of SLAM research based on LiDAR sensors". In: *International Journal of Sensors* 1.1 (2019), p. 1003 (cit. on p. 31).

[Zhe21] K. Zheng. “ROS Navigation Tuning Guide”. In: *Robot Operating System (ROS): The Complete Reference (Volume 6)*. Ed. by A. Koubaa. Cham: Springer International Publishing, 2021, pp. 197–226. ISBN: 978-3-030-75472-3. DOI: 10.1007/978-3-030-75472-3_6. URL: https://doi.org/10.1007/978-3-030-75472-3_6 (cit. on p. 31).

[ZZZ14] T. T. Zhou, D. T. Zhou, A. H. Zhou. *Unmanned drone, robot system for delivering mail, goods, humanoid security, crisis negotiation, mobile payments, smart humanoid mailbox and wearable personal exoskeleton heavy load flying machine*. US Patent App. 14/285,659. Sept. 2014 (cit. on p. 17).

## Declaration