Masters in information technology
Autonomous Intelligent systems
by Prof. Peter Nauth

Localization and Kidnapped Robot Problem - ROS (Group2)

Gaurav Honnavara Manjunath
gaurav.honnavaramanjunath@stud.fra-uas.de

Sandhya Bagadi
sandhya.bagadi@stud.fra-uas.de

***Abstract*— This paper proposes autonomous navigation and detects and solves the Kidnapped Robot Problem in the Turtlebot using the SLAM algorithm. The process, logic, and control of the robot's movement are done using the Robot Operating System (ROS) framework. This framework enables its users to quickly create multi-threaded programs that can communicate via a high-speed TCP/IP network. The robot is simulated in the gazebo, and Rviz is used for data visualization. The Gmapping package is used for mapping by utilizing laser and odometry data from LIDAR sensors. The Turtlebot provides open-source software to perform navigation. MCL (Monte Carlo Localization) performs global localization, meaning that given a map of an environment, it requires no knowledge of the robot's initial position to identify its current position. This is achieved as the robot drives around the environment by observing data and probabilistically estimating the robot's current pose. The method is designed to provide accurate detection across a broad range of particle convergence levels while minimizing reliance on the relocalization/recovery process. The proposed method considers the weight difference between particles, the maximum current weight, and the standard deviation difference between particles. Combining these two parameters for kidnapping detection is considered better than using only the maximum current weight parameter. Six different similarity measures are investigated and compared to particle weight-based detectors to determine how well each detector can distinguish between normal and kidnapping conditions. These simulations show that similarity-based detectors have a higher general accuracy across all kidnapping points than particle weight-based detectors and that relocalization has unaffected accuracy.**

Keywords*— *ROS, SLAM, MCL Kinetic, Rviz, Gazebo, Gmapping, Turtle Bot, Odometry, Kidnapped Robot Problem (KRP), Kidnapping Detection, TCP/IP

I. INTRODUCTION

Autonomous navigation robots must estimate their current location to perform their intended function. In industrial applications, the use of autonomous robots is increasing. Autonomous robots can navigate in locations dangerous to workers with minimum human intervention. Various issues in autonomous navigation, like mapping, localization, and path planning, can be solved using different approaches.

With today's technology, Global Positioning System (GPS) is a popular position estimation method for human and robot navigation, but there are frequent scenarios in which something other than GPS is a viable choice for use with autonomous vehicles. For example, GPS signals cannot penetrate the thick concrete walls of a multilevel structure such as a hospital or school, and signal availability is limited in a downtown environment with a high density of tall buildings. Even if GPS signals are accessible for a robotic application, the GPS position estimates may not be as precise or arrive as frequently as a particular scenario demands. Therefore, autonomous robots require a way to estimate the robot's current position without using a GPS. The solution is provided by the field of robot localization, which develops algorithms that generate real-time estimates of a robot's current position using a map of the robot's environment, data from the robot's interoceptive and exteroceptive sensors, and motion models derived from the robot's kinematics, or mathematical equations based on the robot's structure that describes the way motion forces change the robot's position.

One such problem is the kidnapped robot problem, defined as a condition when the robot is instantly moved to another position without being told during the robot's operation. Detecting a kidnapped robot is one of the most difficult problems in Monte Carlo localization (MCL). This is due to the nature of the particle filter used in MCL itself, where the convergence process of hypotheses (called particles) causes the absence of Particles in some areas, leading to a localization failure if the robot is kidnapped to that area. Kidnapped robot problems do not often happen in practice; however, it is often used to test the ability of the algorithm to recover from global localization failures. Furthermore, the mechanical and sensor faults can lead to a condition

similar to kidnapping. Thus, the detection of this event can be used as fault detection.

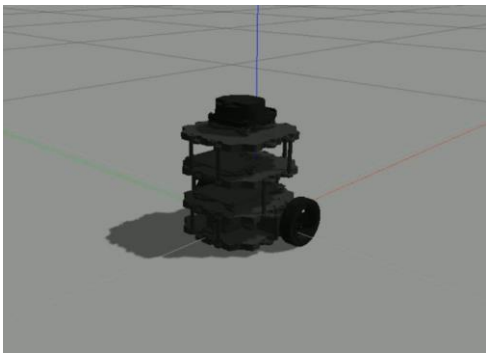

Figure 1 Turtlebot3 in Gazebo

II. RELATED WORK

In Research article [1], the authors have discussed new methods to detect the kidnapped robot problem event in Monte Carlo Localization. The proposed method uses the sensor reading of the robot to determine if the robot's displacement at a particular time instance is considered a natural displacement. A series of simulations are designed to measure detection accuracy and how it compares to other methods. In Research article [2], the authors have presented a solution for the global localization of a Kidnapped robot. The algorithm is based on adapting the 2D SURF (Speeded Up Robust Features) image features to 3D landmarks of the environment. This way, the prior map contains only 3D Surf landmarks. When the robot wakes up in an unknown location, the robot starts to search after these landmarks, which are compared to the priory known ones based on the SURF descriptors. The RANSAC random sampling method was used to estimate the displacement between the online and offline landmarks. This method also describes the advantage of false detection and elimination of associated landmarks. In Article [3], The proposed approach explicitly considers obstacle velocities, Position estimation using an appropriate Kalman-based observer. The velocities are used to predict the obstacle positions within a tentacle-based approach. The implementation consisted of sensors and cameras for Visual navigation following a path as a part of the training phase, also with the lidar sensor to detect/avoid static and moving obstacles, which were not present during training. The authors of paper [4] presented an experimental analysis of GMapping and RTAB-Map regarding their SLAM accuracy, quality of produced maps, and use of produced maps in navigation tasks—the research aimed at ground robots equipped with RGB-D sensors for indoor environments.

III. METHOD FOR KIDNAPPED ROBOT PROBLEM

SLAM algorithm is used for autonomous navigation. SLAM estimates the robot's position by moving it along an unknown area. The quality of the map produced increases with the number of efforts to explore uncharted territory. SLAM algorithm is implemented using the GMapping Tool. GMapping Tool requires odometry data (encoder data from the wheels) and laser data (Kinect data). SLAM Gmapping creates a 2-D occupancy grid map like a building floor plan. The robot software development is done in ROS (Robot Operating System). ROS provides services like message passing between processes and packages, low-level device control, and implementing common functions. In ROS, the process is shown in graph format, and the processing happens in nodes. All the complex 3D simulations are done in GAZEBO. Models, GZ Sever, GZ Client, and GZ Web are the main components of it. GAZEBO has features like built-in robot models, cloud simulation, dynamics simulation, advanced 3D graphics, command line tools, and TCP IP transport. With the help of GAZEBO, we can conceptualize Inertia, Forces, and Sensor Information.

IV. SIMULATION ENVIRONMENT

A. *Robot Operating System (ROS)*

The Robot operating system is an open-source meta-operating system under BSD (Berkeley Software Distribution) license. ROS's built-in tools and libraries can write, build, and test code across multiple computers. ROS provides various services, including low-level device control, implementation of commonly used functionality, message exchange between processes, and package management. ROS allows us to use pre-created packages such as Gmapping and teleop key, which reduces development time. Instead of implementing the entire system in hardware, ROS is used to create a virtual environment, generate a robot model, implement the algorithms, and visualize it in the virtual world. Gazebo and rviz are used to create the virtual environment.

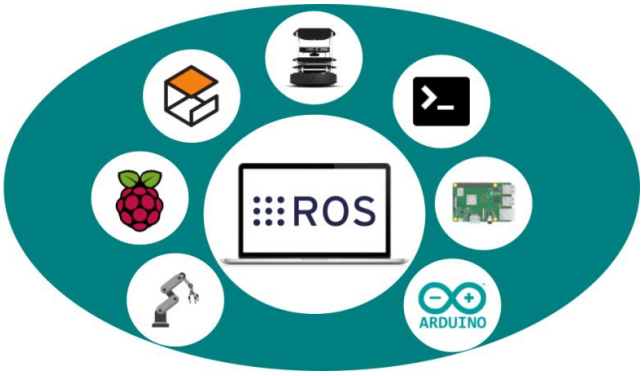

Figure 2 Built-in Tools available in ROS

B. *Gazebo*

Gazebo is an open-source 3D robotics simulator. Gazebo simulated real-world physics in a high-fidelity simulation. It helps developers rapidly test algorithms

and design robots in digital environments. It provides the ability to test the robotic model in the simulated environment and incorporate the data from the sensors. Gazebo allows testing the performance of the robotic model in extreme conditions without damaging the hardware. It uses a physical engine for illumination, inertia, gravity, etc. An XML file called the (URDF) Universal Robotic Description Format is used to describe different elements of the robot. *Figure 1* shows the 3D model of the turtlebot simulated in the gazebo. The robotic model is tested in the environment shown in *Figure 4*.

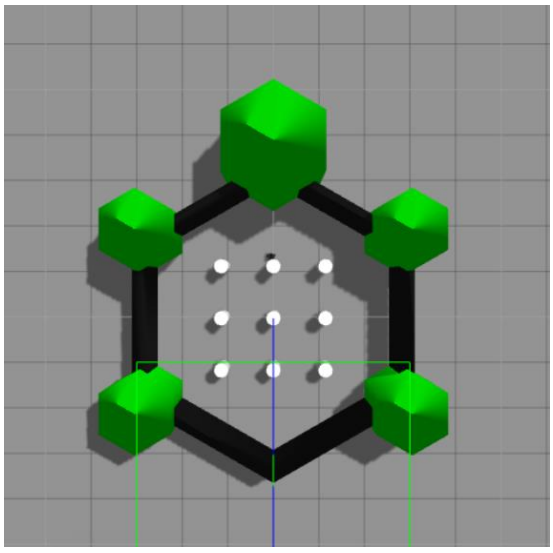

Figure 3 Gazebo Simulation in ROS World

C. *Rviz*

Rviz, abbreviated as ROS visualization, is a powerful 3D visualization tool for ROS: It enables the user to examine a simulated robot model, log sensor data from the robot's sensors, and replay the logged sensor data. The user can debug a robot application from sensor inputs to planned (or unplanned) actions by visualizing what the robot sees, thinks, and does. Rviz displays 3D sensor data from stereo cameras, lasers, Kinects, and other devices in point clouds. Webcam and RGB camera 2D sensor data can be viewed as image data in rviz.

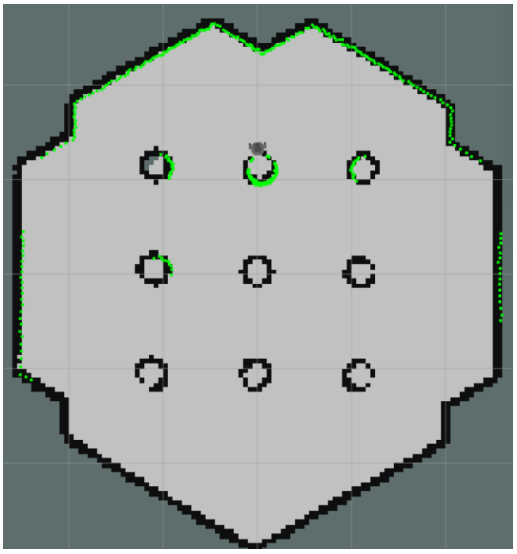

Figure 4 Gmapping Simulation in rviz

V. SLAM ALGORITHM

Autonomous robots can navigate indoors and outdoors without colliding with obstacles in the Environment. Simultaneous localization and mapping is a technique used to make a map, update it, and estimate the robot's position. Some algorithms are used to solve the problem, which include particle filter, Extended Kalman filter, FastSLAM, Covariance intersection, and Graph-based SLAM. The SLAM algorithm combines Mapping, Sensing, Kinematic modeling, Multiple Objects, Multiple Cameras, Moving Objects, Loop closure, Exploration, and Complexity. The main part of SLAM is a range measurement device used to observe the Environment. The range measurement device depends on different variables. The robot uses landmarks to determine its location using measuring devices and sensors. When the robot has sensed a landmark, it extracts the input and identifies the Environment. Some SLAM algorithms exist: CoreSLAM, Gmapping, KartoSLAM, Lago SLAM, and HectorSLAM. In this Project, the SLAM algorithm used is Gmapping which relies on a Particle filter. The Objective of the particle filter is to compute the posterior distributions of the states of a Markov process if there are some noisy and partial observations. Particle filters use approximate prediction and updates.

The samples from the distribution are given in the form of particles, and each particle has a weight assigned to it, representing the probability of particles being sampled. Gmapping is a profoundly proficient Rao-Blackwellized particle filter that creates grid maps from laser data. This package contains a ROS wrapper for OpenSlam's Gmapping. The Gmapping package provides a laser-based SLAM as a ROS node called slam Gmapping. Using slam gmapping, you can create a 2-D occupancy grid map from the LIDAR sensor and pose data the robot collects. [5]

The robot moves, reaching a new point of view of the scene. Due to unavoidable noise a [6]nd errors, this motion increases the uncertainty of the robot's localization. An automated solution requires a mathematical model for this motion. We call this the motion model.

The robot discovers interesting environmental features that must be incorporated into the map. We call these features landmarks. Because of errors in the exteroceptive sensors, the location of these landmarks will be uncertain. Moreover, as the robot's location is already uncertain, these two uncertainties must be composed appropriately. An automated solution requires a mathematical model to determine the position of the landmarks in the scene from the data obtained by the sensors. We call this the inverse observation model.

The robot observes landmarks that had been previously mapped and uses them to correct both its

self-localization and the localization of all landmarks in space. In this case, therefore, both localization and landmarks uncertainties decrease. An automated solution requires a mathematical model to predict the measurement values from the predicted landmark location and the robot localization. We call this the direct observation model.

With these three models plus an estimator engine, we can build an automated solution to SLAM. The estimator is responsible for properly propagating uncertainties each time one of the above-mentioned situations occurs.

Figure 5 SLAM simulation of the surrounding area

VI. MONTE CARLO LOCALIZATION

In mobile robot localization practice, it is almost impossible for a robot to know exactly its coordinates and heading (collectively known as pose) in the given map. Rather, the robot should infer the data from environment. The obtained state is then called belief. The belief of the robot is defined as

$$bel(St) = p(st|z1{:}\,t, u1{:}\,t)$$

This posterior is the probability distribution over the state $s_t = \langle x_t, y_t, \theta_t \rangle$ at time *t*, given all past measurements $Z_{1:t} = \{z_{1,} z_{2,} \dots\dots\dots z_{t,}\}$ and all past controls $u_{1:t} = \{u_{2,} \dots\dots\dots u_{t,}\}$ Sometimes it is also useful to consider the belief *before* taking the current measurement, that is

$$bel(st) = p(st|z1{:}\,t-1, u1{:}\,t)$$

MCL is a Bayes-based localization algorithm. It provides a powerful tool to calculate posterior bel(∗), given measurement and control data, Bayes filter is based on Markov world assumption, that is, past and future data are independent if one knows the current state s_t.[1] By implementing Bayes rule and this Markov world assumption, the belief posterior can be defined as

$$bel(st) = \eta p(zt|st)\, bel(st)$$

The term $p(s_t|\, s_{t-1} u_{1:t})$ is defined as the prediction or motion model, since it reflects the state transition due to robot motion. The probability $p(z_t|\, s_t)$ itself is called correction or sensor model, since it incorporates sensor reading to update robot state. η is the normalization constant ensuring the result to be normalized to one. Bayes filter gives freedom to the choices of representation for the posterior. MCL represents the posterior $bel(st)$ by a set St of N weighted samples distributed according to the posterior. The density of the samples proportionally represents the likelihood of the robot's pose being there.

$$St = \langle s_t^{[n]}, \omega_t^{[n]} \rangle; \quad n = 1,2, \dots, N$$

Each particle $s_t^{[n]}$ represents the hypothesis of the robot's pose at time *t*. The $\omega_t^{[n]}$ is the nonnegative number called weight of particle. It indicates how good particle $s_t^{[n]}$ in representing the robot's pose.

A key issue with particle filters is keeping up random distribution of particles all through the state space, which gets complicated if the problem is of high magnitude. Because of these reasons, it is preferred to utilize an adaptive particle filter which merges a lot quicker and is computationally considerably more effective than a primitive particle filter. MCL is a probabilistic localization system used in robots moving in 2D. It performs the Monte Carlo localization approach, which utilizes a particle filter to follow the position of a robot against a previously known map. The first step to utilizing an adaptive particle filter for localization is to generate a map of the environment. The robot can be set to a random location or start from no initial estimate of its position. After the robot moves forward, new samples start that will predict the robot's position after the command. Random uniformly distributed samples can be added as the robot recovers if its losses track of its position.

VII. IMPLEMENTATION & RESULT

A. *Hardware*

TurtleBot3 Burger - TurtleBot3 is a ROS-based mobile robot that is compact, inexpensive, and programmable for education, research, hobby, and product prototyping. The purpose of TurtleBot3 is to drastically reduce the platform's size and cut the price without sacrificing functionality or quality while allowing for future expansion. Depending on how you reconstruct the mechanical pieces and employ optional parts like the computer and sensor, the TurtleBot3 can be customized

in various ways. The TurtleBot3 uses SLAM, Navigation, and Manipulation as its fundamental technologies, making it ideal for home care robots. The TurtleBot can construct a map and drive around your room using SLAM (simultaneous localization and mapping) algorithms.

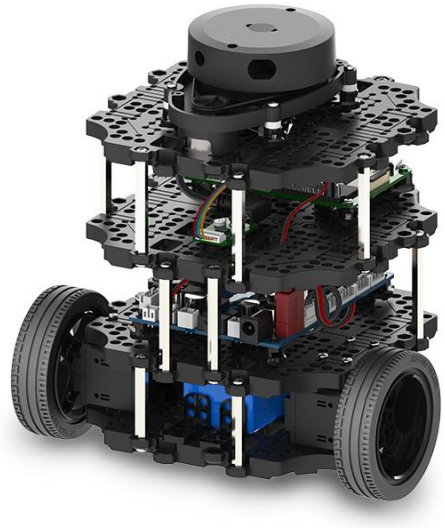

Figure 6 Turtlebot 3 Burger

Lidar - Laser distance sensors are optical distance-measuring devices. They can measure the distance to a given object using a non-invasive laser beam.

Laser distance meters can also be proximity sensors to detect objects within a specific range.

There exist a few ways of measuring the distance, such as:

- Time-of-Flight principle
- Triangulation and similar techniques

The time-of-flight principle is the most common form of operation. In this mode of operation, a pulse of the laser beam is sent toward the measuring object. This beam is narrowed down using a lens system.

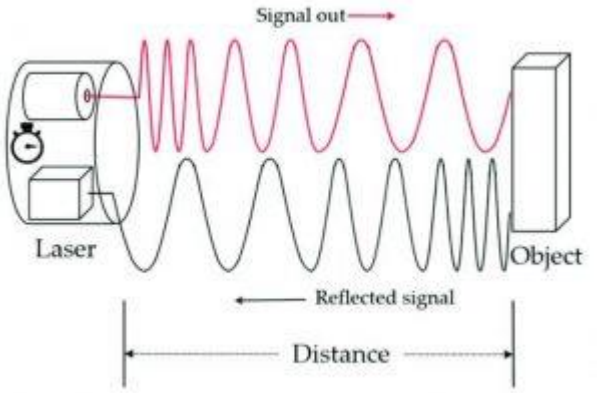

Figure 7 Principle of Lidar Sensor

When the beam hits the object, it is reflected off the surface toward the sensor. The sensor measures the time the beam takes to travel to the target and return.

The LDS-01 is a 2D laser scanner capable of sensing 360 degrees that collect data around the robot for SLAM (Simultaneous Localization and Mapping). The LDS-01 is used for TurtleBot3 Burger, Waffle, and Waffle Pi models.

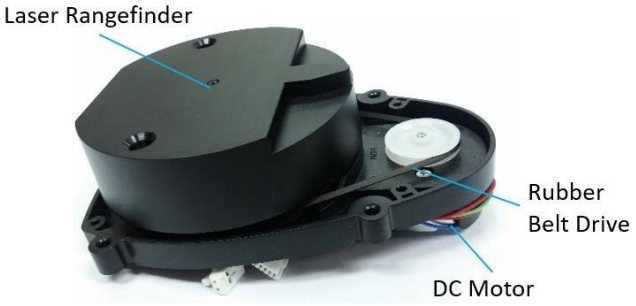

Figure 8 Lidar Sensor - LDS 01

Raspberry Pi 3 Model B+ - The Raspberry Pi is a low-cost, **credit-card-sized computer** plugs into a computer monitor or TV and uses a standard keyboard and mouse. It is a little-capable device that enables people of all ages to explore computing and learn programming in languages like Scratch and Python. It can do everything you'd expect a desktop computer to do, from browsing the internet and playing high-definition videos to making spreadsheets, word processing, and playing games.

Raspberry Pi 3 Model B+ is the latest Raspberry Pi 3 range product, boasting a 64-bit quad-core processor running at 1.4GHz, dual-band 2.4GHz and 5GHz wireless LAN, Bluetooth 4.2/BLE, faster Ethernet, and PoE capability via a separate PoE HAT.

Figure 9 Raspberry pi 3 b+ Model

OpenCR 1.0 - OpenCR1.0 (Open-source Control module for ROS) is an open-source robot controller embedded with a powerful MCU from the ARM Cortex-M7 line-up.

It Supports RS-485 and TTL to control the Dynamixels and offers UART, CAN, and a variety of other communication environments. Development tools such as Arduino IDE are available as well. It has the advantage of being able to operate more powerfully when used with a host controller such as an SBC (Single

Board Computer). It provides various exclusive sources based on ROS to maximize the functions of OpenCR1.0 when using ROS.

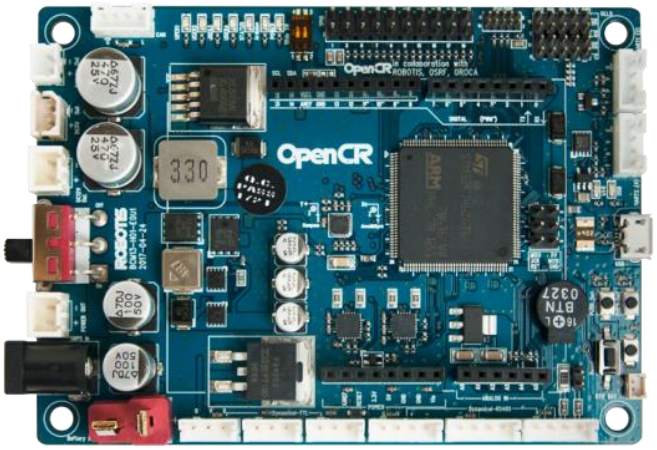

Figure 10 OpenCR 1.0

B. ***Software***

Linux Machine - Unix/Linux Machine It is the machine required for all heavy computations. The robot sends data from its sensors to the computer. The information is fed into MCL by the computer. Simultaneously, a movement command is delivered from the computer to the robot.

The Linux Machine corresponds to the Remote PC, which will control TurtleBot3. Ubuntu 16.04.7 LTS (Xenial Xerus) is used for Simulating the turtle bot 3 burger (Kinetic)

ROS - The Robot Operating System (ROS) is a message-passing service that makes robotics easier. Every system component (sensors and actuators) is run by individual nodes that handle the low-level software that runs the component. We can then send messages to run the SLAM, Teleoperation, Navigation, and Position Estimation nodes.

C. ***Project architecture***

Catkin - Catkin is a ROS-customized compile and building system extended from CMake.

Catkin workspace - Catkin workspace is the folder that manages all the packages. It is compiled using catkin. The 'src' folder contains source code of packages. The 'build' folder contains a cache of Catkin/CMake and middle files for compiling. The 'dev' folder contains target files, say, head files, link libraries and executable files. All the work and programmes are located in the 'src' folder.

Package - The package contains one or multiple executable files(nodes). A package always contains a 'CMakeLists.txt' file and a 'package.xml' file. The 'CMakeLists.txt' file defines Catkin compile rules. The 'package.xml' file defines the attributes of the package. The package contains launch file (in 'launch' folder) and yaml file(in 'config' folder). Launch file allows multiple nodes to execute multiple nodes at the same time.

D. ***Communication framework***

Master – the node manager - Each node registers at master before it starts. The master manages the communication between nodes. we need to enter 'roscore' every time we start ROS because it starts master. (REMOTE PC)

Node – process in ROS - Node is a live instance of the executable file of a package. Using 'rosrun [pkg_name] [node_name]' to start a node. Using 'rosnode list' to list information of all current running nodes. Using 'roslaunch [pkg_name] [file_name.launch]' to start master and multiple nodes at the same time. The launch file defines the rule for nodes to start.

E. ***Communication methods***

ROS mainly has Topic, Service, Parameter Service and Actionlib as its communication methods. Topic Using predefined topics to communicate between nodes. It uses the publish-subscribe method. Topic acts as a channel that the nodes can publish and subscribe. Topic is an asynchronous method. Message is the data type defined for the topic. It is stored in '.msg' file. It uses synchronous request-reply method. It only acts when it is requested. Service is very useful in occasionally called tasks. Multiple clients request one server. Srv is the data type defined for service. It is stored in '*.srv' file. Parameter Server It maintains a dictionary that stores parameters. Action A server-like communication method with status feedback. It is usually used in time-consuming and preemptive tasks. Action is the data type defined for Action. It is stored in '.action' file.

F. ***Map Generation***

The map generation is carried out by first generating the environment and importing turtlebot in Gazebo. The robot model consists of two wheels and the Lidar sensor is mounted on to the frame. The mapping process using the Gmapping package. The sensor captures the depth image of the environment using Infrared sensors and acts as a rangefinder device. This depth image is converted into a 3d point cloud using the depth image proc package in ROS. Further this 3d point cloud data is converted into 2d laserscan using point cloud to laserscan package. By providing all the required parameters to the packages in Gmapping a map is created in rviz. The *Figure 11* below shows the generated map. Initially the robot is moved in the simulated environment using the teleop key package in ROS by sending velocity commands via keyboard. The *Figure 12* shows the rqt graph of Gmapping. The Gmapping package provides /map topic which can be

used by the map server package to save the generated map. The map is now ready to be utilized for autonomous navigation.

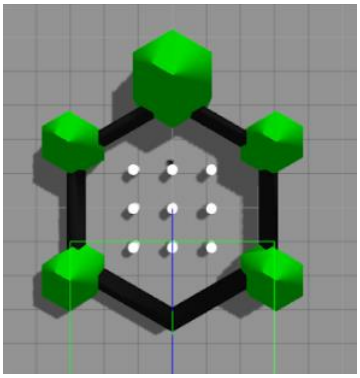

Figure 13 Turtlebot3 Gazebo World Simulation

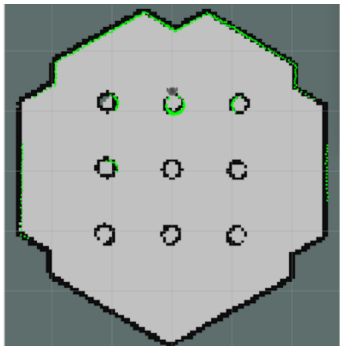

Figure 14 Turtlebot3 rqt graph of Gmapping of turtlebot world

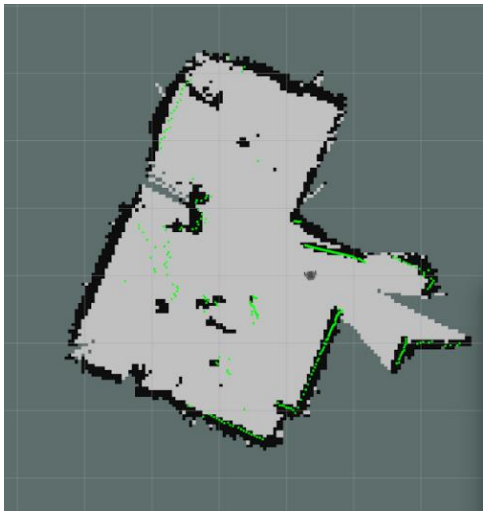

Figure 15Turtlebot3 rqt graph of Gmapping of turtlebot real world

G. ***Localization***

Localization estimates the position and pose of the robot with respect to the environment. To localize the robot laser data, odometry data and the map of environment are essential. The turlebot uses MCL (Monte Carlo Localization) for localization. It follows a probabilistic localization approach for robots in 2 dimensions. In MCL the pose of the robot is tracked against the known map of the environment using a particle filter. To reduce the computational requirement KLD (Kullback–Leibler divergence) sampling is utilized to Fig. 3: Visualizing map in rviz automatically adapt for sample size. The “Fig. 4” shows the self-localized robot rviz.

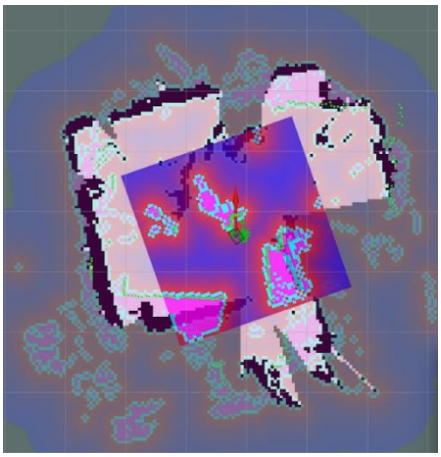

Figure 16 Autonomous Visualization in rviz

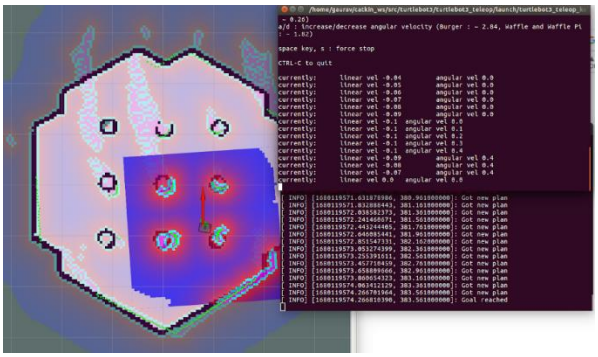

Figure 17 Turtlebot3 Visualization and teleop mode

H. ***Autonomous navigation***

The autonomous navigation can be performed once the mapping and localization is completed. The ROS navigation stack packages are used to implement AMCL. The node to perform localization on a static map is provided by ROS AMCL package. This node subscribes to the TF data,2D laser scan data provided by the robot and the previously generated static map. The pose of the robot and its estimated position with respect to the map is published by the AMCL node. The Rviz visualizes the global and local costmap. The robot moves autonomously. The 2D nav goal in Rviz is used to assign a destination goal to the robot in the map along with its orientation. The path to the desired destination is planned by the robot and velocity commands are given to the robot controller. The *Figure 18* shows the path followed by the robot to navigate from its initial position to the desired destination goal assigned.

VIII. CONCLUSION

In conclusion, The Kidnapped Robot Problem is a critical issue that autonomous robots face when their initial pose is unknown or when they are unexpectedly relocated. Various techniques, such as particle filters and Kalman filters, have been developed to estimate the robot's pose with high accuracy.

The Kidnapped Robot Problem, localization, and SLAM are critical topics in robotics that require continued research and development. With the help of advanced algorithms and platforms like Turtlebot3, we can enhance the capabilities of autonomous robots and enable them to operate more efficiently and effectively in real-world scenarios.
We carried out an experimental analysis on an autonomous navigation system using ROS. We used turtlebot3 burger and Linux Machine (Ubuntu 16.04) to simulate, test and validate the project.

With the help of the Robot Operating system(ROS), we Simulated Localization which helped the robot determine its position and orientation in the Gazebo and real-world environment, while SLAM enabled the creation of maps of the surroundings as the robot moved through unknown environments. We validated autonomous navigation and position estimation using the generated maps from the simulation (Gazebo, rviz).

Overall, this project's findings demonstrate the effectiveness of the Robotic Operating System as a platform for experimentation and various algorithms, such as SLAM, Localization, MCL, which helps in position estimation and autonomous navigation.

IX. REFERENCE

[1] F. S. a. F. C. Andrea Cherubini, "Autonomous Visual Navigation and Laser-based Moving Obstacle Avoidance," *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS.*

[2] R. S. X. a. L. M. G. G. Bruno M. F. da Silva, "Mapping and Navigation for Indoor Robots under ROS: An Experimental Analysis," *Research Gate* , 2019 July.

[3] A. M. E. M. P. E. L. T. E. I. S. P. D. E. G. L. T. U. o. C.-N. R. Eng., "New approach in solving the kidnapped robot problem," *41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*, 2010.

[4] Z. H. I. Iksan Bukhori, "Detection of kidnapped robot problem," *International Journal of Advanced Robotic Systems* , pp. 1-6, 2017, July-August.

[5] J. Sol`a, "Simulataneous localization and mapping with the extended Kalman filter," Januart 17 2013 .

[6] ROBOTIS, "ROBOTIS e-Manual," [Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/.

[7] N. Feng, "Laser Distance Sensor," October 2021. [Online]. Available: https://www.omch.co/laser-distance-sensor/#:~:text=Laser%20distance%20sensors%2C%20also%20known,objects%20within%20a%20certain%20range.

[8] M. M. P. P. V. T. P. k. Sumegh Pramod Thale, "ROS based SLAM implementation for Autonomous navigation using Turtlebot," *International Conference on Automation, Computing and Communication*, vol. 32, no. ITM Web Conf, p. 2020.

[9] E. Brennan, "The Kidnapped Robot Problem as a Classification Problem: Using Artificial Neural Networks to Characterize Robot Localization," *Saint Louis University ProQuest Dissertations Publishing,* no. 13885072, 2019.

[10] C. Y. a. B.-U. Choi, "Detection and Recovery for Kidnapped-Robot Problem Using Measurement Entropy," *Communications in Computer and Information Science book series* , vol. 261.