Implementing Autonomous Navigation on an Omni Wheeled Robot Using 2D LiDAR, Tracking Camera and ROS

Atharva Bhorpe[1], Pratik Padalkar[2], and Pawan Kadam[3]

[1,2,3]Pimpri Chinchwad College of Engineering, Pune
bhorpeatharva06@gmail.com[1]
padalkarpratik19@gmail.com[2]
pawankadam1107@gmail.com[3]

Abstract. This paper demonstrates the implementation and results of autonomous navigation algorithms on an Omni Wheel-based Robot using ROS (Robot Operating System). The basis of this application is autonomous navigation of the robot using Simultaneous Localization and Mapping (SLAM), specifically GMapping and Autonomous Path Planning Algorithms. The actual robot is Arduino-based, equipped with a tracking camera for Odometry data and a 2D LiDAR sensor for laser scan data of the environment. The robot is built on Omni wheels, making it possible to perform holonomic movements. The results of tuning the autonomous algorithms for this holonomic robot are also presented.

Keywords: Tracking Camera · LiDAR · ROS · GMapping · SLAM · RViz · tf · Odometry · Arduino · TEB Local Planner · AMCL.

1 Introduction

1.1 Introduction

We are entering a new era of Industry 4.0. Robotics and automation are important aspects of it. Robotics is the multidisciplinary branch of engineering of building devices that physically interact with their environment. Robots are used in automobile industries, medical institutes, food processing, which perform simple tasks, and in industries where work must be performed in environments hazardous to humans. But advances in technology have encouraged many researchers to develop more intelligent and adaptable robots. These robots should have sufficient intelligence to behave sensibly for a long time, whilst achieving specific tasks. Industries that require mobile robots are shifting towards autonomous robots which helps them to do the task efficiently. Almost every industry is adopting such robots which perform their tasks on their own. These autonomous robots are helping industries to revolutionize faster.

Omni Directional Motion In omnidirectional motion, the robot can travel in every direction under any orientation. The robot's movement can be categorized in 3D space, specifically x, y (position of the robot in the 2D plane) and ω (orientation of the robot in Z-axis). This motion system can give linear and angular velocities simultaneously. An omnidirectional mobile robot is a type of holonomic robot. It can move independently and simultaneously in translation and rotation. The rollers in Omni wheels help to reduce the friction of the robot and help to drag in a certain direction[6].

Holonomic Motion System Holonomic is the relationship between controllable and total degrees of freedom of a robot. The robot achieves the Holonomic motion when total degrees of freedom are equal to the controllable degrees of freedom. The robot used for this paper has three degrees of freedom. It has three Omni wheels attached at an angle of 120°, which makes the robot holonomic. In autonomous robots, holonomic movement is very useful as this motion helps achieve the goal faster, and changes the robot's orientation when it can't reach the goal by holonomic movements[6].

Inverse Kinematics Robot kinematics is the detailed mathematical and analytical study of the motion of a robot. The robot kinematics is classified into two parts: forward kinematics and inverse kinematics. Forward kinematics calculates the position of the end effector from manually configured values for the joint parameters. Inverse kinematics is just the opposite to forward kinematics. Inverse kinematics uses kinematic equations to determine the robot's motion to reach the desired goal. Inverse kinematics takes the cartesian, effector position, and orientation as input and calculates the joint's angles. Inverse Kinematics is used to calculate and trace the trajectory of the robot's links taking into account the joints and degrees of freedom of the robot[6].

1.2 Robot Operating System

ROS is an abbreviation for Robot Operating System. It is an open-source robotics meta operating system. It has a set of software libraries and tools that help us to build any complex robotic application. It uses the nodes, topics, packages, messages, and services for communication between robots and the ROS framework. Multiple nodes, topics, and messages can be published simultaneously in ROS. All this data is handled by *roscore*. Using ROS we can control many robots simultaneously. ROS can be easily implemented using Python, C++, and Java.[1] ROS Noetic is selected for the application[1].

RViz RViz is the 3D visualization software used to visualize robots in ROS. It provides a convenient graphical user interface to visualize sensor data, robot models, environment maps, which helps in developing and debugging your robot controllers[3].

Launch File This file is of the format *.launch* and uses XML language format. It provides users a convenient way to run multiple ROS nodes from a single file and provide a way to modify different parameters[1].

2 Robot Base Description

Computer

- CPU → Intel Core i5 (8th Gen) 8300H / 2.3 GHz
- RAM → 8GB DDR4 SDRAM
- Memory → 128GB Kingston SATA SSD
- Operating System → Ubuntu 20.04

Microcontroller Arduino Mega 2560 microcontroller is used on this robot for controlling all actuators. This microcontroller is connected to the ROS server through serial communication.

Wheels Omni wheels of diameter 100 mm are selected. We used three Omni wheels placed at 120° apart from each other in a triangular pattern. These wheels can travel in 2 directions and rotate independently.

Motors Three RS775 DC motors of 680 RPM and Torque 40 kg-cm are used to control the three Omni wheels for locomotion.

Battery LiPo 4 cell of 8000mAh is used to power the motor drivers.

Motor Drivers Three BTS motor drivers supply 43A of current to drive the three navigation motors. This motor driver is selected because it is suitable to handle the required payload.

Tracking Camera We have used Intel® RealSense™ Tracking Camera T265 for getting a robot's Odometry data. ROS has official support for the Intel RealSense Tracking Camera and has its official package to use this camera called *realsense2_camera*. This package creates a ROS node to subscribe the robot's Odometry data from the camera's topic and publishes it over other ROS topics, to use in autonomous navigation.

LiDAR (Light Detection and Ranging) We have used the RPLiDAR A1M8 sensor on our robot, which is a low-cost LiDAR sensor suitable for SLAM applications. The sensor is used for getting the robot's laser scan data. It can give up to a 360° field of view for laser scan and a 12-meter range with an update frequency of 7Hz. ROS has its official package to use this laser sensor called *rplidar_ros*. This package creates a ROS node that publishes laser scan data over a topic to use it for robot's localization[11].

3 System Architecture

3.1 ROS Navigation Stack

The ROS Navigation Stack is a set of packages of environment mapping and path planning algorithms for autonomous navigation. On a core level, the Navigation Stack takes data from Odometry, Sensor streams, and processes the data taking into account the goals, obstacle information, and output velocity commands to the robot[8].
The Navigation Stack requires the below algorithms to setup autonomous navigation for a robot:

Mapping Mapping is a process that creates a 2D plan of the robot's environment. SLAM(Simultaneous Localization And Mapping) is one of the techniques used to perfectly Map the unknown environments and simultaneously localize as well. The generated map is used for path planning and location estimation. We used the GMapping algorithm for our environment mapping[11]. It is a highly efficient Rao-Blackwellized particle filter-based algorithm that provides laser-based SLAM.
GMapping acts as a ROS node and takes data from both laser sensor (LiDAR) as well as from robot pose (Tracking camera) and creates a 2D grid map of the environment. *Fig.8* shows the map built by the robot using GMapping.
This map can be visualized in RViz and retrieved via a ROS topic or service. To save this map we need the ROS *map_server* package which runs *map_server* node that reads the map and saves it to the local computer storage respectively. The map created is stored in two formats: *.yaml* and *.pgm*. The YAML file contains the map meta-data and the image file name, whereas the image file encodes the occupancy data, and contains the actual map[9].

Localization Localization is the process of estimating the location of the robot with respect to its environment. We have used Adaptive Monte Carlo Localization (AMCL) algorithm for carrying out robot localization. AMCL is a probabilistic localization system. It uses a particle filter to estimate the difference in laser scans and track the pose of the robot against a known map[10]. This algorithm takes the map, laser scan readings, and TF messages as input, and outputs estimated pose[10]. It initializes a particle filter taking into account the parameters in the configuration file on start-up.

Path Planning Path planning is the process of autonomously creating paths for the robot to follow. There are several algorithms to accomplish this for visualization. We need two-path planners: Global and Local Planners. Global planner computes the optimum path taking into account the static obstacles from the map[7]. Local planner revises the calculations of the path to avoid dynamic obstacles in close proximity[7].

Global Planner For this Planner,the Dijkstra algorithm is selected. Dijkstra uses the information of a grid cell map and converts the problem into a graphic search method, which provides the shortest path for navigation[7].

Local Planner The selected Local Planner is Timed Elastic Band (TEB) which consists of recalculating the initial global plan taking into account the kinematic model of the mobile base and providing the updated path considering the dynamic obstacles and possible deviations in the path[7].

Move Base The move_base node is the most important in the navigation stack. It contains the global & local planners, their parameters configuration, costmap computation. This node takes in the sensor source data, Odometry data, map data, localization data and outputs the velocity commands to reach the specified goals safely and efficiently[4, 7].

All the above algorithms work together according to the graph shown below in *Fig.1*

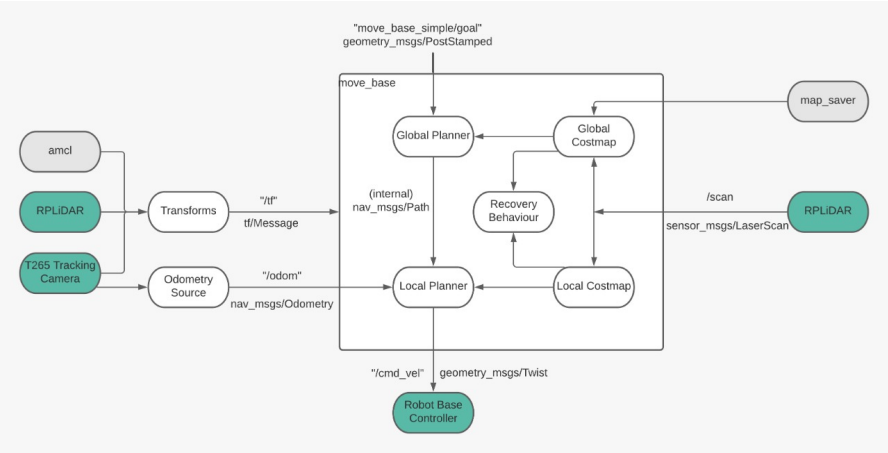

Fig.1 Navigation Stack block diagram

3.2 ROS-Arduino Communication

We have used an Arduino microcontroller to control our robot and pass the commands from the ROS server to an Arduino using ROS serial to the robot wheel. ROS has official support to connect an Arduino to ROS and has its official package called *rosserial*. This is the metapackage which includes different *rosserial* packages. We have used the *rosserial_python* package on PC side. This package creates a ROS node called *serial_node* that publishes the linear and angular speed data through the *cmd_vel* topic and an Arduino subscribe to this topic and sends commands to the motors connected to the wheels accordingly. Refer *Fig.2*.

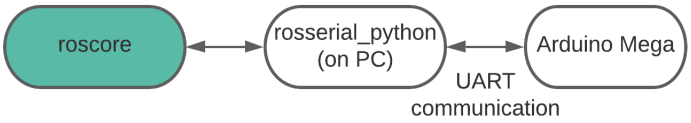

Fig.2 Rosserial Flow Diagram

We have to specify the serial port to which the Arduino board is connected to the computer. We can also set the baud rate in the launch file as per the requirement, this baud rate must be the same as the one mentioned in the Arduino code.

3.3 Robot Control

Keyboard To control the robot using a keyboard *teleop_keyboard.launch* file is launched, in which the *teleop_twist_keyboard* package is included. This package publishes the robot's total speed through *cmd_vel* to roscore. An Arduino subscribes to this *cmd_vel* topic to send commands to the motors.

Dual Shock 4 Controller DS4 is one of those controllers with which we can control any kind of robot easily. We connected DS4 to the computer in wireless mode.
The *ds4drv* helps in connecting DS4 to the computer wirelessly using Bluetooth. The *joy_node* connects to the Controller, reads values of all keys and joysticks, and publishes their values at topic */joy* to *roscore*. Refer *Fig.3*.

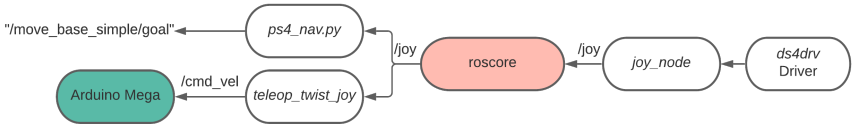

Fig.3 DS4 flow diagram

Foxglove Studio (ROSBridge) ROSBridge is a web socket server with a JSON (JavaScript Object Notation) API exposing the ROS services and publisher, subscriber functionality[12]. It is the ROS package that enables two-way communication between ROS server and web app.
The launch file includes the ROS node to set up the rosbridge connection between the ROS server and the web application which we will be using for the controlling of the robot.
We have used a web application called Foxglove Studio which works on the rosbridge web socket server. We connected Foxglove to the ROS server using rosbridge. Then we set up the connection between ROS and UI of the app through which we can easily control our robot[2]. Refer *Fig.4*

Fig.4 Foxglove flow diagram

3.4 ROS Mobile

ROS mobile is an open-source Android application developed by some ROS open source contributors. This app mainly focuses on dynamic control and visualization of any kind of robot in a ROS environment that works as a ROS node initializing publishers and subscribers with ROS messages. To control the motion of the robot using the ROS mobile app we have set up the connection between the ROS server and the ROS mobile application using the IP address of roscore[5].

4 Experimental Set Up

4.1 Setting up the Robot Base

Fig.5 shows the Omni Wheel based Robot Base, with RPLiDAR mounted on the top, Tracking Camera mounted in front of the LiDAR, and Computer placed behind the LiDAR.

Fig.5 Robot Base

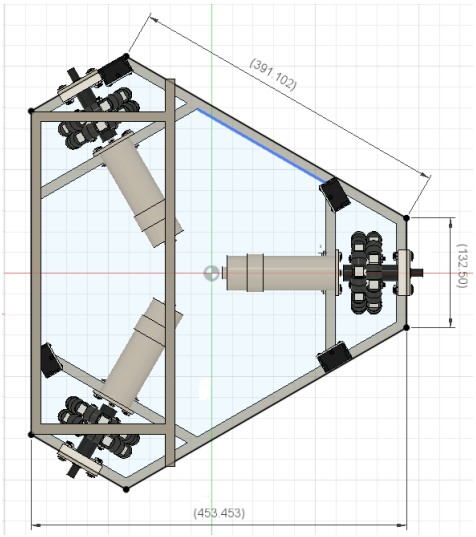

Fig.6 Robot Base Dimensions (in mm.)

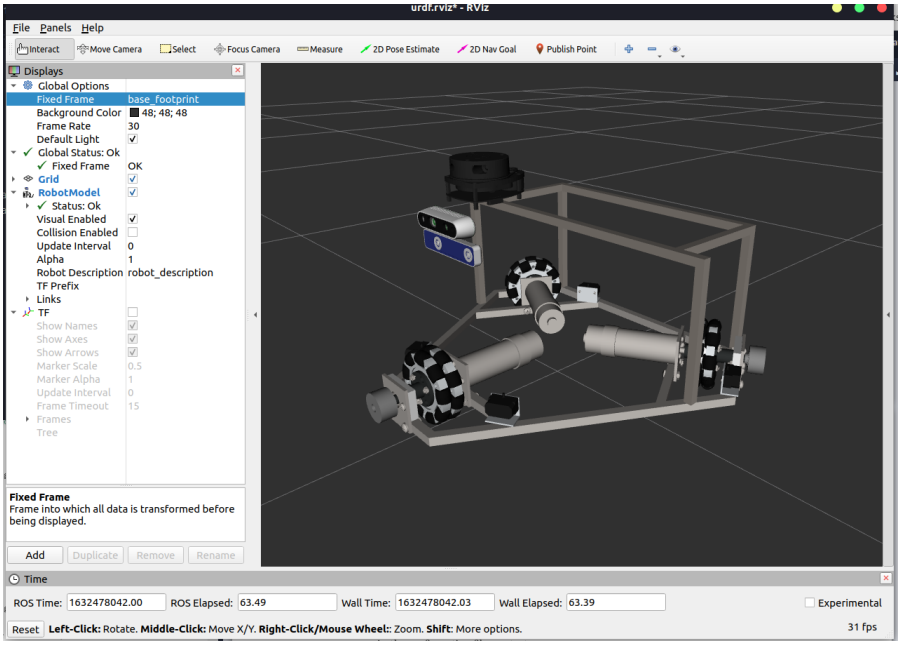

Fig.7 RViz Window displaying the Robot Base

Inverse Kinematics Equations As shown in *Fig.8* we have used a three-wheel Omni robot with three wheels which are 120° apart from each other placed in a triangular pattern[6].

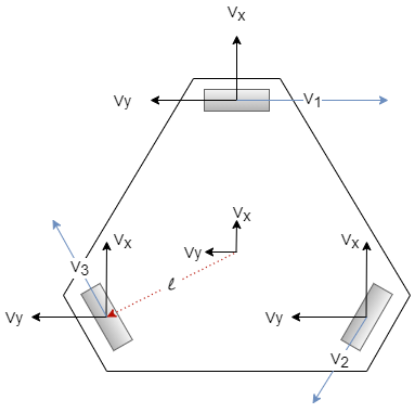

Fig.8 Inverse Kinematic Model

Consider the distance from the center of the robot to the wheel is l. The linear velocities of all the three wheels v_1, v_2, v_3 in x and y directions are v_{1x}, v_{2x}, v_{3x}, and v_{1y}, v_{2y}, v_{3y} respectively. v_x and v_y be the velocity of the robot in x and y directions respectively. v_z be the angular velocity of the robot.

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \frac{1}{r} \begin{bmatrix} -l & 1 & 0 \\ -l & -1/2 & -sin(\pi/2) \\ -l & -1/2 & sin(\pi/2) \end{bmatrix} \begin{bmatrix} \omega_z \\ v_x \\ v_y \end{bmatrix} \tag{1}$$

The inverse kinematic equation of the robot are as follows :

$$v_1 = 0v_x + 0.67v_y + 0.33\omega_z \tag{2}$$

$$v_2 = -0.58v_x - 0.33v_y + 0.33\omega_z \tag{3}$$

$$v_3 = 0.58v_x - 0.33v_y + 0.33\omega_z \tag{4}$$

These are the three equations that we have used in Arduino code which makes the robot movement holonomic as well as non-holonomic.

4.2 Running the Code

The ROS nodes launched are given below in the order of launching:

```
$ roslaunch so-bot_bringup hardware.launch
$ roslaunch so-bot_navigation so-bot_navigation_teb.launch
```

The first command launches rosserial, rplidar, and tracking camera nodes. The second command launches *map_server*, *amcl*, *move_base* and RViz nodes(*Fig.9*).

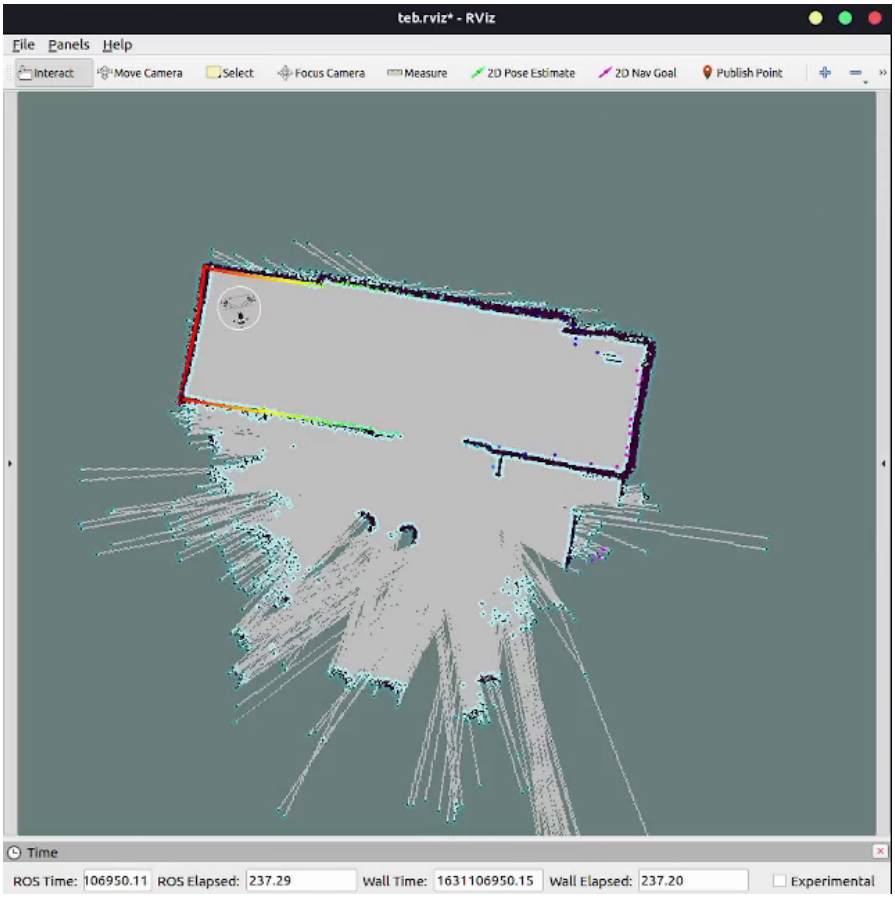

Fig.9 Rviz Window showing the Map for Autonomous Navigation

4.3 Providing Destination Co-ordinates for Autonomous Navigation

Using DS4 To control the robot by DS4 we need to run the python script:

```
$ rosrun so-bot_navigation ps4_nav.py
```

This python script contains the code for autonomous movement of the robot from one starting point to another point coordinates. One publisher and one subscriber are added to this code. Subscriber first subscribes to the joy node when buttons of DS4 are pressed and then publishes a point on the topic move_base_simple/goal.

Using Foxglove Studio We connected this app to the ROS server using rosbridge. The coordinates of some points on the map are stored in custom buttons in the Studio. To make the robot travel to the desired location, click the button containing the coordinates of that location, and the robot will try to reach that point autonomously(*Fig.10*).

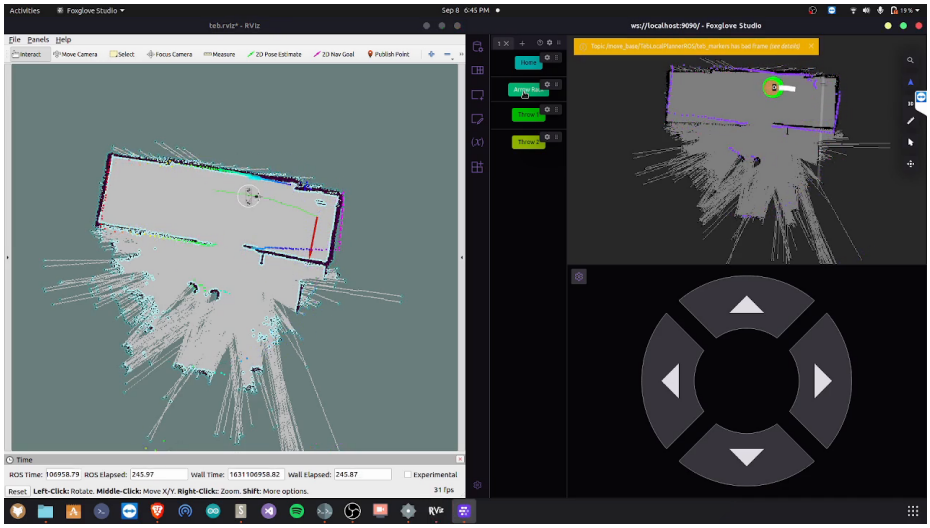

Fig.10 Autonomous Navigation Goal using Foxglove Studio(right)

After the coordinates of the destination point are published it is subscribed by Global Planner. Global planner which uses Dijkstra's algorithm then finds the shortest path. With the help of this path and odometry, the Local planner calculates the required velocity for moving the robot to its destination. Both planners also subscribe to laser scan data which helps them to change the velocity accordingly if any obstacles are detected on the map. This calculated velocity is then subscribed by the Arduino with the help of rosserial communication. This entire algorithm helps the robot to reach the destination autonomously.

5 Results

From this study, it was observed that the quality and specifications of the sensors used are deterministic factors on the quality of autonomous navigation planning and execution.
The local paths that TEB Planner creates are in arcs with high curvature, which makes the locomotion smooth, but increases the navigation time between the start and end coordinates.
A comparison of the total time taken to travel between two fixed points using the different navigation methods is shown in the *Table 1* below:

Navigation Method	Approx. Time taken (sec)
Manual Non-Holonomic	15
Autonomous Non-Holonomic	11
Manual Holonomic	10
Autonomous Holonomic	9

Table.1 Approx. Time to Travel using Different Navigation Methods

The Autonomous Holonomic Navigation took the least time to move between a starting point and the destination point.

6 Conclusion

In this paper, we have developed an autonomous holonomic robot. Using Tracking Camera, we have carried out odometry. GMapping SLAM procedure is used to build an incremental map using a LiDAR sensor. Finally, we carried out localization and navigation of the robot in the map we built. We used move_base to configure global and local planner parameters specific for this robot for optimum path planning and locomotion.

References

1. ROS Wiki, https://wiki.ros.org. Last accessed 7 Sep 2021
2. Foxgolve, https://foxglove.dev. Last accessed 7 Sep 2021
3. RViz, http://wiki.ros.org/rviz. Last accessed 7 Sep 2021
4. move_base, http://wiki.ros.org/move_base. Last accessed 7 Sep 2021
5. Rottmann, Nils and Studt, Nico and Ernst, Floris and Rueckert, Elmar.: "ROS-Mobile An Android application for the Robot Operating System," arXiv preprint arXiv, (2020).
6. Riky Tri Yunardi, Deny Arifianto, Farhan Bachtiar, Jihan Intan Prananingrum.: "Holonomic Implementation of Three Wheels Omnidirectional Mobile Robot using DC Motors," Journal of Robotics and Control(JRC), Volume 2, Issue 2, (2021). https://doi.org/10.18196/jrc.2254
7. Pablo Marin-Plaza , Ahmed Hussein , David Martin, and Arturo de la Escalera.: "Global and Local Path Planning Study in a ROS-Based Research Platform for Autonomous Vehicles." Journal of Advanced Transportation, Volume 2018, pp. 1–10. (2018). https://doi.org/10.1155/2018/6392697
8. Li Zhi, Mei Xuesong.: "Navigation and Control System of Mobile Robot Based on ROS," IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference(IAEAC 2018). pp. 368–372. (2018). https://doi.org/10.1109/IAEAC.2018.8577901
9. Yassin Abdelrasoul, Abu Bakar Sayuti HM Saman, Patrick Sebastian.: "A Quantitative Study of Tuning ROS GMapping Parameters and Their Effect on Performing Indoor 2D SLAM," 2016 2nd IEEE International Symposium on Robotics and Manufacturing Automation (ROMA). pp. 1–6. (2017). https://doi.org/10.1109/ROMA.2016.7847825
10. Fitria Romadhona Quratul Aini, Agung Nugroho Jati, Unang Sunarya.: "A Study of Monte Carlo Localization on Robot Operating System," 2016 International Conference on Information Technology Systems and Innovation (ICITSI). pp. 1–6. (2017). https://doi.org/10.1109/ICITSI.2016.7858235
11. Zhang Xuexi, Lu Guokun, Fu Genping, Xu Dongliang, Liang Shiliu.: "SLAM Algorithm Analysis of Mobile Robot Based on Lidar," 2019 Chinese Control Conference (CCC). pp. 4739–4745 (2019). https://doi.org/10.23919/ChiCC.2019.8866200
12. Christopher Crick, Graylin Jay, Sarah Osentoski, Odest Chadwicke Jenkins.: "ROS and Rosbridge," 2012 7th ACM/IEEE International Conference on Human-Robot Interaction (HRI). pp. 493–494 (2012). https://doi.org/10.1145/2157689.2157846