Autonomous Mobile Vehicle Using ROS2 and 2D-Lidar and SLAM Navigation

Pratham Gyanani

gyprathamgmail.com

Medi-Caps University

Mridul Agarwal

Medi-Caps University

Ranjeet Osari

Medi-Caps University

Garima Chandore

Medi-Caps University

Research Article

Keywords: Robot Operating System (ROS), Adaptive Monte Carlo Localization (AMCL), SLAM Toolbox, Simultaneous localization and mapping (SLAM), URDF, Odometry, ROS2 Control, Rviz, Gazebo

Posted Date: May 3rd, 2024

DOI: https://doi.org/10.21203/rs.3.rs-4323431/v1

Additional Declarations: No competing interests reported.

Autonomous Mobile Vehicle Using ROS2 and 2D-Lidar and SLAM Navigation

Author - Pratham Gyanani[1] ,
Co-Author - Mridul Agarwal[2] ,
Contributing Author - Ranjeet Osari[3] ,Garima Chandore[4]
Dept. of Computer Science Engg.
Medi-Caps University, Indore
gypratham@gmail.com [1], agarwalmridul020@gmail.com [2], ranjeet.osari@medicaps.ac.in [3],
garima.chandore@medicaps.ac.in [4]

***Abstract*— This study describes the use of the SLAM algorithm in autonomous mobile vehicle/robot navigation. The proposed work is organized inside the Robot Operating System (ROS2) architecture. Data visualization is done with Rviz, and vehicle simulation is done with Gazebo. SLAM Toolbox uses laser and odometry data from several sensors for mapping. We have built a small bot which uses Differential Drive Control System. It also uses the Nav2 plugin of ROS2 for navigation from one place to another in the environment by first giving the initial co-ordinates of the robot and then the Goal pose where the robot has to reach. It also implements online asynchronous plugin of ros2 which helps the robot to detect real time change in the map during the navigation so it will change its path if a new object is detected real time**

***Keywords*—Robot Operating System (ROS), Adaptive Monte Carlo Localization (AMCL), SLAM Toolbox, Simultaneous localization and mapping (SLAM), URDF, Odometry, ROS2 Control, Rviz, Gazebo.**

I. INTRODUCTION

In recent years, there has been remarkable progress in the field of autonomous navigation, particularly within outdoor agricultural environments. This advancement holds immense promise for revolutionizing agricultural practices by substantially reducing labor costs through the integration of autonomous robotic vehicles. These vehicles emulate human-like navigation capabilities, offers a viable solution to labor-intensive tasks. Key challenges that we have faced is that, for autonomous navigation it requires a map, current location of the robot(localization), and plan a path, which all are addressed through various approaches. This work seeks to explain how to develop and simulate a robot that can visually recognize and avoid stationary objects using Simultaneous Localization and Mapping (SLAM) techniques while independently navigating to predetermined destinations. Ackerman steering systems, which are skilled at maneuvering through complex agricultural terrains, are used to assist effective mobility in such difficult environments. So we first implement the URDF[1] file for our robot which is base of our robot design, than we have to implement the drive control system which helps the robot to understands its hardware so that it can interact with the easily and can control them effectively for precise movement in the field. Than by using SLAM algorithm for generating a 2-D map using 2D Lidar sensor data which than further use in for the navigation purpose. So to do all these a Framework is required which can integrate all these things together, so this is where ROS comes in handy which provides a great work environment for the development of robots and understands how hardware and software works together effectively.

II. LITERATURE REVIEW

The writers of this research study have explored several approaches to SLAM implementation using the Extended Kalman filter and Rao-Blackwellized filter which helps them understand the data coming from the lidar can be collected and used to generate a map. Autonomous navigation has been included by the authors of article [1]. The Author evaluates several SLAM algorithms, including CoreSLAM, Gmapping, and Hector SLAM [2]. Researchers use Lidar sensor and Rviz implements the autonomous navigation on TurtleBot [2]. The authors [3] of have provided a brief explanation of how to use an RF ID sensor for object level mapping in an interior setting [4]. In this paper, the authors[5] are checking the flexibility of a SLAM based mobile robot to map and navigate in an indoor environment. It is based on the Robot Operating System (ROS) framework. The model robot is made using gazebo package and simulated in Rviz. The mapping process is done by using the GMapping algorithm, which is an open source algorithm. The aim of the paper is to evaluate the mapping, localization, and navigation of a mobile robotic model in an unknown environment. The authors of the research paper [6], made a navigation platform with the use of automated vision and navigation framework, With the use of ROS, the open source GMapping bundle was used for Simultaneous Localization and Mapping (SLAM). Using this setup with rviz, the turtlebot 2 is implemented. Using a Kinect sensor in place of laser range finder, the cost is reduced. A research paper [7] compares 3 SLAM algorithms core SLAM, Gmapping, and Hector SLAM using simulation. The best algorithm is used to test unmanned ground vehicles(UGV) in different terrains for defense missions. Using simulation experiments they compared the performance of different algorithms and made a robotic platform which performs localization and mapping.

III. RELATED METHODOLOGY

First to design a robot which can move easily in the environment like a agriculture field, because it's not like the

robots design that work in industries now days as they all work on the concrete roads or tiled roads, where there is not much problem of turning the robot or to gain speed at a constant pace. In a agriculture field where the path is all bumpy so the robot also has the capability to maintain its centre of gravity so that it will not fall to one side and also the turning in the field is not simple like on a road the friction is more than the road so much more force is required to turn the wheels while maintaining the balance and not deviate from the path it is following. To simplify the task of designing the robot URDF we use Xacro files in which multiple files can be used to determine a single a robot by each file having a feature of the robot. Fig. 1

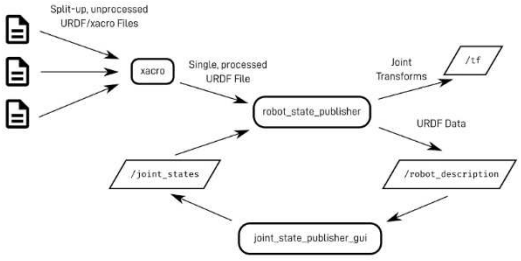

Fig.1 Multiple Xacro file into a Single URDF

Then identifying a drive control system which matches best with our problem statement so we have used Ackerman steering system which is a 4 Wheel Drive and 4 Wheel steering system. And to implement the drive control system, using ROS2_control which provides many more different drive controls like differential drive, bicycle steering, tricycle steering systems etc. Fig.2 is an idea how our robot would look like.

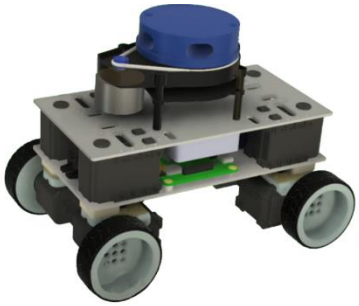

Fig. 2 Robot Visualization in Gazebo

SLAM (Simultaneous Localization and Mapping) is concurrent localization and mapping, aims to let a robot or other autonomous vehicle map a foreign environment while concurrently figuring out where the robot is in relation to the area. Although there are many different mapping and localization techniques available, simultaneous mapping and localization adds complexity to SLAM.

The SLAM algorithm is utilized in autonomous navigation implementation. By operating the robot through the unknown region, SLAM is utilized to determine the location of the robot in the field. The quality of the map generated increases with the number of efforts to explore the unknown area. SLAM Toolbox is used to implement SLAM algorithms. Odometry (data of encoders from the Wheels) and laser (Kinect) data are needed by SLAM Toolbox. To produce a 2-D habitation gird map, similar to floor plan of a building, use Slam toolbox function.

Robot Operating System, or ROS, is used for developing robot software. ROS offers features including low level device control, common function implementation, and message transfer between packages and processes. In ROS, processing takes place in nodes and is shown in a graph structure.

In GAZEBO, all of the intricate 3D simulations are completed. Its primary elements are GZ web, GZ server, GZ client, and models. Pre-Built models of robot, cloud and dynamic simulation, powerful visuals in 3D, TCP/IP transport and command line tools are only a few of GAZEBO's features. We can comprehend forces, inertia, and sensor data with the aid of GAZEBO.

IV. ANALYSIS OF ROS2, 2D-LIDAR AND SLAM

A. ROS:

Under a Berkeley Software Distribution (BSD) license, Robot Operating System (ROS) is an open-source meta operating system.

With the pre-built tools, assets and libraries that ROS offers, code may be written, developed, and tested on a variety of systems. Many services offered by ROS allow us to use pre-existing packages like those that have already been produced. These services include managing low level machines, providing commonly used services, passing information between various operations, and managing collections. Teleop key and slam toolbox cut down on development time.

In this work, communications are done in structure of topics that are published among various nodes using ROS. Instead of constructing the entire system on the hardware itself, ROS can also use to generate virtual environments, produce robot models, implements algorithms, and visualize it in the simulated environment. Using Gazebo and Rviz, the virtual environment is created.

ROS provides a great way to design and develop the robot, it has 2 Versions ROS and ROS2 and both of them has some Distributions also for this we have used ROS2-FOXY, which has the end-of-life of April 2023.

The difference between ROS and ROS2 is that, ROS is developed for robust traditional applications which has defined features, specially for the educational and research purpose. However ROS2 is developed for high performance systems, which can perform better in real-time, and distributed systems. Both have their own specific use as per the project you're working on. Both works on same principle of publishing and receiving topics through nodes.

B. ROS2-Foxy:

A significant release for the open-source robotics community, ROS 2 Foxy Fitzroy raises the bar for production robot development. Technology enhancements, improved quality, characteristics, resources, and improved communities' protocols are all introduced. The latest release of ROS 2 contains features such as rolling releases, security policies, and new community methods for determining quality levels. It also prioritizes enhanced security features, security monitoring, and the establishment of a safe environment for industrial robots. ROS 2 Foxy Fitzroy

includes critical functionality, automated processes, and supporting mechanisms to let developers create production-ready robotics devices and services.

C. 2D Lidar:

A 2D LiDAR sensor is an active optical sensor that scans and identifies surfaces in the vicinity of robots. These sensors use Time-of-Flight (ToF) technology to compute the distance among objects and itself through measuring the amount of time it takes light to go across space and return. 2D LiDAR sensors are widely used in a variety of industries, including optical navigation, robotics, automotive systems, presence detection, and medical technology. They provide two-dimensional measurement of distance by sensing values sequentially at comparable periods of time. These sensors are critical for creating exact maps of the environment, allowing robots to move and interact with their surroundings. For sensing 2D lidar data, we used X4-Ydlidar and the gazebo laser plugin in a simulation environment.

V. PROBLEM DEFINATION AND SOLUTION

- Manually Mapping the Environment
- To find the vehicle's location on the created map.
- Using the online asynchronous approach for live detection of any items that may be in the route.
- Manipulating the robot autonomously.

VI. SIMULATION

A. Gazebo :

Gazebo, a 3D dynamic simulator, is used to simulate complex robotic models in both indoor and outdoor environments.It provides the ability to integrate sensor data and evaluate the robotic model in a simulated environment. With Gazebo, testing the robotic model's performance in harsh environments is possible without endangering any mechanics. It generates effects like as gravity and illumination using a physical engine. The Universal Robotic Description Format (URDF), an XML file, is used to describe the robot's numerous components. "Fig. 2" shows the 3D model of the robot simulation in the gazebo.The settings depicted in "Fig. 3" and also how 2D lidar sensor would be looks in gazebo environment to test the robotic model. Gazebo helps to simulate the real environment by using built-in design or by importing .stl file for your environment , so that you can assess all the parameters for your real world implementation.

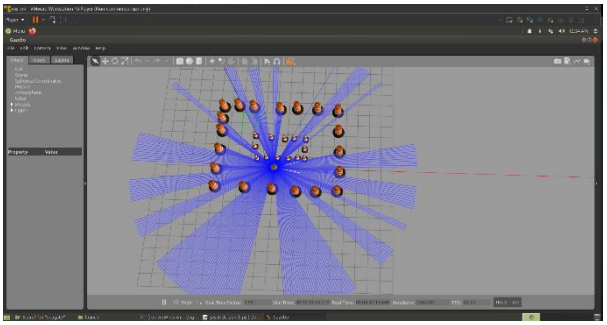

Fig. 3 Gazebo environment

B. Rviz :

RViz, the Robot Operating System (ROS) visualization tool, offers a powerful 3D interface for understanding robot behavior. It functions as a multi-faceted window, allowing users to delve into various aspects of a robot's operation.
One key capability of RViz is the ability to revisit past experiences. By replaying recorded sensor data, users can gain insights into the robot's performance in different scenarios. This is similar to rewinding a video to examine certain actions or reactions.

Another useful feature is the viewing of the simulated robot model. This virtual representation provides a platform for observing the robot's motions and interactions with its surroundings. It's like having a digital twin of the robot to explore with in a safe and controlled environment.

Furthermore, RViz allows users to directly inject sensor data. This capability allows for the testing and debugging of robot actions using simulated sensor inputs. It's like providing the robot hypothetical sensory information to see how it reacts in different situations.

By leveraging these capabilities, users may effectively debug robot applications. RViz enables them to view the robot's whole decision-making process, from raw sensor data perception to planned actions and unanticipated replies. This complete picture is critical for finding and correcting flaws with the robot's programming.

Visualizing the Spectrum of Robot Senses RViz does not restrict itself to a particular type of sensor data. It supports both 3D and 2D sensor data, resulting in a varied visualization experience.

RViz produces point clouds from 3D sensors such as Kinects, stereo cameras, and lasers. These point clouds provide a precise image of the robot's surroundings, much like a 3D map created from sensor data.

In contrast, RViz displays 2D sensor data from webcams and RGB cameras as images. This allows humans to view exactly what the robot's cameras are capturing, providing useful information on the robot's visual perception.

In essence, RViz functions as a strong translator, converting raw sensor data into an understandable visual format. This visual representation is useful for understanding a robot's inner workings and assuring its proper operation. Figure 4 shows the Rviz work environment.

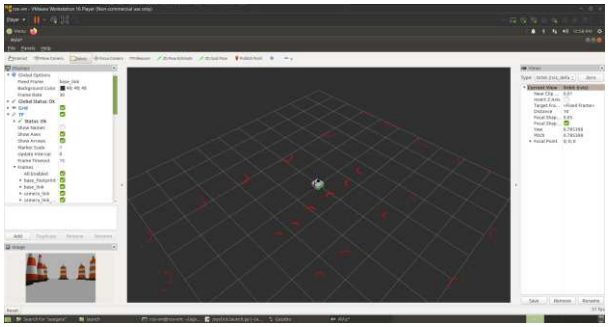

Fig.4 Rviz

VII. ALGORITHMS FOR SLAM

Robots that are autonomous can move around both indoor and outdoor spaces without running into any obstacles. The approach known as "simultaneous localization and mapping" is used to create and update maps as well as estimate robot position. The particle filter, Extended Kalman filter, FastSLAM, Covariance intersection, and graph-based SLAM are a few of the techniques that are used to solve it. Mapping, sensing, kinematic modeling, multiple objects, multiple cameras, moving objects, loop closure, exploration, and complexity are all combined to form the SLAM algorithm. The primary component of SLAM is a range measurement device that is used to observe the environment. The range measurement tool is dependent on various factors.

The robot employs sensors and measurement instruments to pinpoint its location based on landmarks. The robot extracts the input and recognizes the surroundings after detecting a landmark. Examples of SLAM algorithms include CoreSLAM, Gmapping, KartoSLAM, LagoSLAM, and HectorSLAM. This article uses the SLAM Toolbox approach, which relies on lidar scan matching. The SLAM Toolbox supports Simultaneous Localization and Mapping (SLAM), which allows robots to move freely by constructing maps of their surroundings. The procedure consists of two major steps: mapping and localization.

Mapping: To precisely map large industrial environments, the SLAM Toolbox employs sensors such as encoders, cameras, IMUs, and laser scanners. It supports both pure localization and a number of mapping methods, including synchronous and asynchronous mapping. By running mapping and localization simultaneously, synchronous mapping maintains a measurement buffer for enhanced SLAM processing.

Asynchronous mapping ensures real-time performance by processing current measurements after the previous measurement is completed. To increase localization quality, pure localization compares measurements from current and previous sessions.

Logic of Map Generation: The SLAM Toolbox creates exact maps by combining incoming sensor data with odometry input to establish the robot's true location. It ensures precise mapping by integrating odometry and sensor data to constantly update the robot's position. SLAM Toolbox, which operates on the ROS 2 platform, is a leading open-source SLAM vendor. In actual applications, the SLAM Toolbox has been demonstrated to be capable of mapping environments as large as 24,000 square meters in real time.

In conclusion, ROS 2's SLAM Toolbox is an effective tool that makes precise maps of environments using sensor data and odometry, allowing robots to travel around freely. The SLAM algorithm is used in many different applications, including autonomous vehicles, unmanned aerial vehicles, mine exploration, and underwater reef monitoring. ROS is an open-source program used to implement SLAM techniques.

VIII. ADAPTIVE MONTE CARLO LOCALIZATION

One of the main challenges for particle filters is to keep an irregular distribution of particles over the state space; this gets increasingly challenging when handling huge problems. Because of these advantages over ordinary particle filters in terms of merging speed and computational efficiency, the use of an adaptive particle filter is recommended.

Multifaceted robots use an AMCL deterministic locating system. It performs the adaptive monte carlo localization method, which uses particle filtering to track a robot's movement versus a previous-known map.

Making an environment map is the first stage in using an adaptive particle filter for localization. The robot may begin at an unknown location or be programmed to start at a random spot. The robot advances, and then a fresh sample is generated to anticipate its position in response to a command. If the robot loses position, random evenly dispersed samples can be added while it recovers.

IX. PROPOSED IMPLEMENTATION

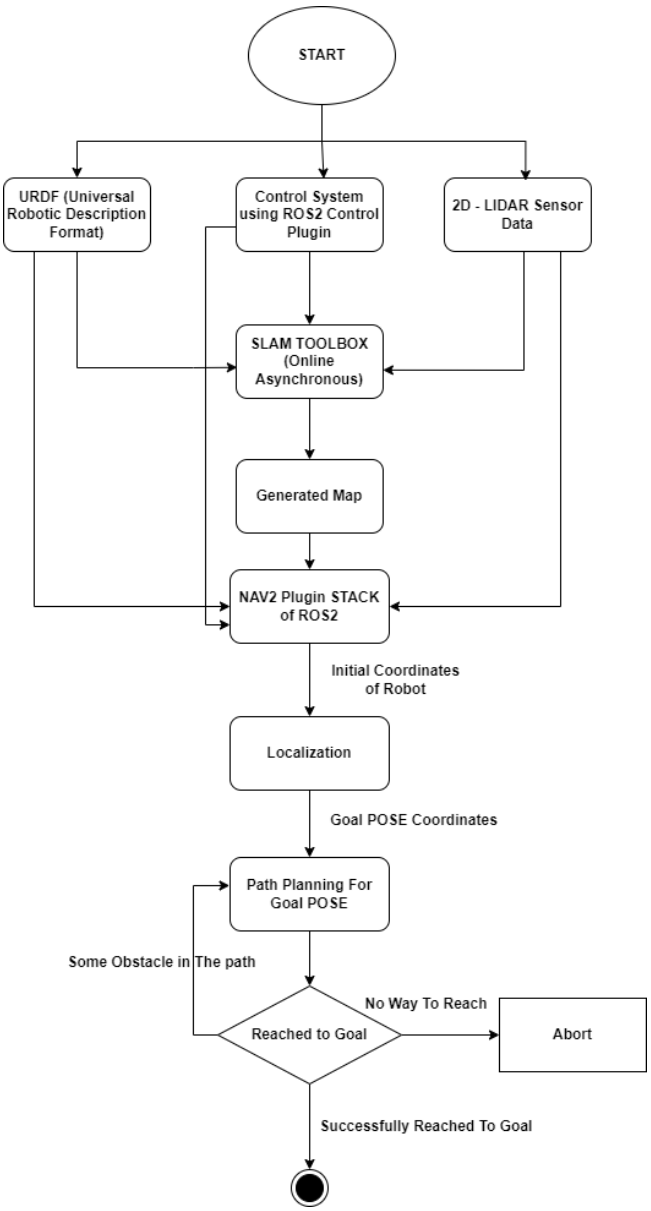

Fig.5 Proposed Implementation

A. Steps involved in implementation -

For Start we required 3 things to proceed with the implementation URDF file, Control System, 2-D Lidar sensor data.

1. **URDF –** It plays an important role even before the navigation process starts. The physical components of the robot are described in this XML file, which includes:

   Links: These stand in for the robot's inflexible components, such as its base, wheels, limbs, etc.
   Joints: These indicate the types of movement (rotation, translation) that are permitted at each joint and how the connections are connected to one another.
   Visual Properties: These specify how the robot looks in a simulation or visualization.
   For a precise physics simulation, the inertial properties of each link—its mass, center of mass, and moments of inertia—must be specified.

2. **Control System (with plugins for ROS 2)-**The software framework known as ROS 2 (Robot Operating System 2) was created especially for robots. It gives programmers the tools they need to build modular robot programs. Consider it as a common language used to communicate amongst various robot sections.

   To analyze sensor data and provide movement commands to the robot, the control system makes use of specialized software components known as plugins that are integrated into the ROS 2 framework.

3. **2D-LIDAR Sensor Data -** A LIDAR sensor functions similarly to the eyes of the robot. It releases bursts of light and clocks how long it takes for the light to return after hitting nearby objects. With the use of this information, the robot is able to create a two-dimensional map of its environment, or a digital depiction of the area it is traveling.

4. **Online Asynchronous SLAM Toolbox -**Simultaneous Localization and Mapping is referred to as SLAM. The system's toolbox is an essential component. It utilizes the received LIDAR data to

   Create a map: As the robot moves (online SLAM), the toolbox creates a real-time map by analyzing the sensor data to comprehend the environment's layout.

   Locate the robot: In order to pinpoint the robot's location on the map, it also evaluates the sensor data (localization). This is analogous to the robot locating itself on the map it is creating.

5. **ROS2 Generated Map -** The SLAM toolbox, upon processing the LIDAR data, creates a two-dimensional map of the robot's surroundings inside the ROS 2 framework. The robot's environment is depicted on this map, complete with walls, obstacles, and possibly even open areas.

6. **NAV2 Plugin Stack:** Another collection of ROS 2 plugins made especially for robot navigation is called the Navigation Stack (NAV2). It employs pathfinding techniques using the created map as input to.

   Make a route plan: The NAV2 plugin stack determines the optimal path for the robot to travel in order to arrive at its destination based on the beginning location and the intended goal position. This route avoids collisions by taking obstructions into account.

7. **Initial Coordinates -** The robot's initial coordinates, or starting point, are defined within the map. This indicates to the navigation system the starting point of the robot's voyage.

8. **Goal Pose Coordinates -** The robot's intended location, or goal pose, is specified. This provides the navigation system with the robot's required location. The robot's intended location and orientation at the destination are included in the goal posture.

9. **Path Planning for Achieving the Goal -** This is where things become magical. The NAV2 plugin stack analyzes the map by taking the objective pose and the starting location into account using complex algorithms. It determines the best route for the robot to take in order to get to its objective while avoiding obstacles based on this data.

10. **Robot Follows the Planned Path -** The NAV2 stack provides the planned path to the control system. After that, it converts this path into a string of movement instructions (such as advance or turn left) and transmits them to the robot's motors. To move around the environment, the robot carries out these commands.

11. **Objective Successfully Attained -**The robot's sensors track its advancement constantly. The navigation process is completed if the robot successfully achieves its objective position, or destination.

12. **Object Detection while navigation -** As the robot navigates, its sensors may pick up unforeseen obstructions along the way that weren't depicted in the original map. In certain situations, the control system halts the navigation and initiates safety procedures to prevent crashes. Depending on the circumstances, the robot may stop or make evasive movements.

13. **Abort (Unreachable Goal) -** Occasionally, obstacles may prevent the robot from taking a path that would allow it to reach the goal position, according to the NAV2 stack. In the event that this occurs, the navigation procedure is terminated, and the goal location or control system may need to be changed.

A. Understanding SLAM and its use

SLAM is used when we have no GPS available, means that we don’t know where robot is in the surrounding. So, using

SLAM as we move in the environment, we keep track of object in our surrounding & their position and also our own position compare to them as we move around & we can also use stride length to help us with the pose estimation and then whenever we see new object in surrounding, so we know where it at because we know where we are. Resultant Map maybe not accurate as GPS, but it will be better than no map at all. SLAM is majorly categorized in two categories-

1. Feature SLAM: In this it learns through some features or landmarks of its surroundings, Feature may also be very small, e.g. the branch of the grape vine.
2. Grid SLAM: In this, it divides the world we see in cells & each cell can either be occupied or un-occupied or somewhere in between.

B. Understanding some Co-ordinate Transformation

We know that a frame link which is attached to our robot is called Base Link. For testing, when we drive the robot around, we set fixed frame in Rviz to odom (odometry) frame, which is the reference frame through which we as world origin if no other reference is provided. The difference between the base link and odom is calculated using wheel drive control odometry.
When robot start, it will set to (0,0), both frames lie on top of each other & as the robot drives around, base link will move compared to odom. There is an important note, that the motion of base link relative to odom may not be exactly correct but it will be smooth. It means that if the robot moves 2m forward in real life it will move 1.9m ahead of odom in Rviz. This error did not appear suddenly, it built up over time, as the odometry drifted. We can take a small section of that trajectory & it would be nice and smooth & basically correct but it just slowly got away from the truth, this is because odometry is effectively a measurement of the Robot's velocity even if it's technically measuring wheel angular position. This velocity is then integrated smoothly over time to produce the position of the robot in Rviz.

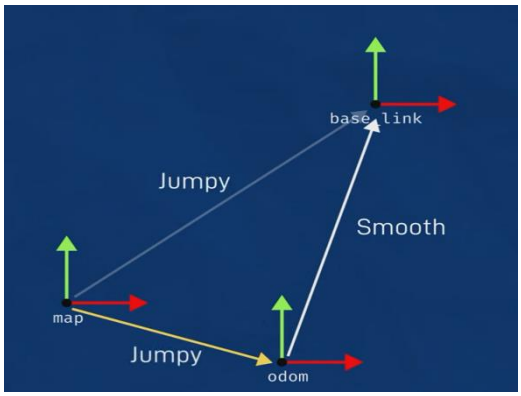

Fig.6 Co-ordinate transformation

C. How SLAM use this Co-ordinate Transform

In order to address the drift in the orbital frame, a correction mechanism such as GPS or SLAM—which measure the robot's position directly—must be implemented. The robot might jump if we use this to update the odom to base link transform. While this might not be a big deal, some algorithms might not function well with this solution.
Since ROS can only have one parent, we must solve the problem by taking the pose estimate from SLAM along with the current odom to base link transform and calculating the appropriate map to odom transform. As a result, we get map frame to base link with a relative pose estimation from map to odom. Alternatively, we use a new frame called the Map frame, so that we can now express the location of the base link compared to map frame. How do the map frame and odom frame share subjects in Figure 6.

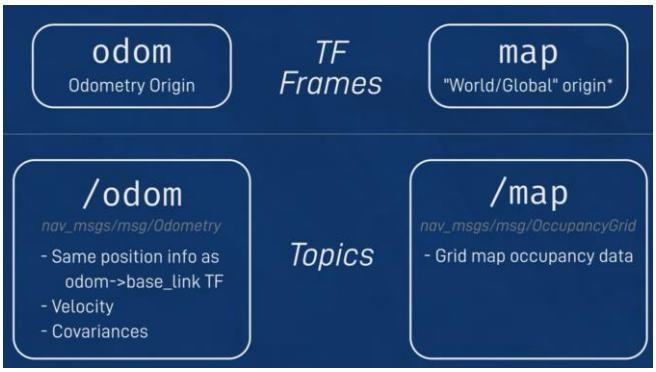

Fig.7 Topics Published During SLAM

D. Map Generation

Creating a Navigation Map for Robots: This section outlines how to build a map for robot navigation in simulation. Here's the gist:

Setting Up the Environment: We start by creating a virtual world in Gazebo and placing a robot within it. A 2D Lidar sensor mounted on the robot acts as the key tool for map generation.

Capturing the Environment: The 2D Lidar sensor uses infrared to capture depth information, like a range finder. The data is initially a array of depth point in 360 degree which tells how much that point is far away from the sensor.

Data Transformation: To use this data for mapping, we need some transformations. ROS packages come to the rescue:

- SLAM Toolbox is used and a method called Online-Asynchronous mapping which is used to generate map and save it offline in the system that can be used afterwards in the navigation
- Pointcloud_to_laserscan transforms the point cloud into a 2D laser scan format preferred by the map generation software.

Building the Map: The Slam toolbox package is the heart of this process. It requires two things from the robot:

- **Odometry data:** This tells gmapping how the robot is moving (position, orientation, changes over time).
- **Laser scan data:** The converted 2D laser scan data (from the 2D lidar) provides the environment details for map creation.

With this information, slam toolbox builds a map of the surroundings. This map can be visualized in RViz (a ROS visualization tool).

Controlling and Visualizing: Initially, the teleop_key package allows manual robot movement using keyboard controls to explore the environment for mapping. Figure 7 shows the corresponding Rviz graph for Grid Slam.

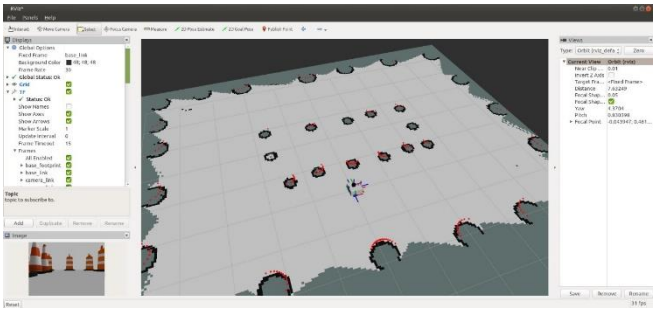

Fig 8 Generated Map

Saving the Map: The gmapping package provides a /map topic that broadcasts the generated map. The map_server package can use this topic to save the map permanently. This saved map becomes the foundation for future autonomous robot navigation, eliminating the need to rebuild it each time.

In essence, this approach utilizes ROS tools and a 2D Lidar sensor within a simulated environment to create a map, enabling robots to navigate their surroundings autonomously.

E. Localization (Finding the position of robot in the map)

The job of locating a robot in its environment and identifying where it is currently and what its pose (position) is known as localization. This procedure depends on three essential components which we used in our implementation:

Laser Data: It is a 360-degree data of the sensor where it returns an array of distance of objects that are around the robot. Tracking robot's movement (position and orientation changes) using odometry data which we get with the help of drive wheel system.

The robot position is then determined on the map using localization.

Adaptive Monte Carlo Localization, or AMCL, is used to get the robot position. In a 2D simulated environment, this probabilistic method is used to tracks the robot's position with respect to a known map that we have generated by SLAM and using particle filter it will calculate the precise location of the robot on the map. AMCL uses KLD sampling, which automatically modifies the number of particles required for precise localization, to maximize computational efficiency (Figure 8).

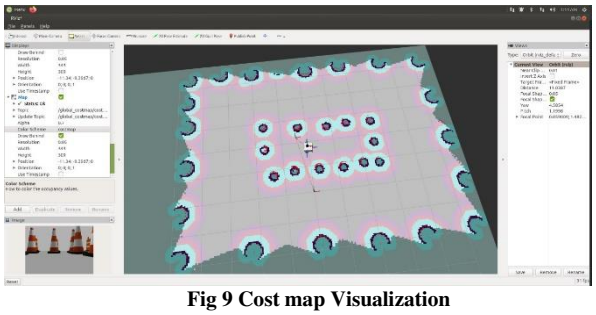

Fig 9 Cost map Visualization

F. Rqt graph

The Rqt graph GUI plugin is part of the Rqt tool package in ROS. The rqt graph is an instrument for viewing all of the currently active nodes and operations along with their interactions. The rqt graph provides an overall perspective of the system. Rqt can be employed as a system debugging instrument. In fig. 9 Rqt graph is generated of our robot

G. Autonomous Navigation

After mapping and localization are finished, autonomous navigation can be carried out. Navigation requires majorly two things –

1. Robot Position Estimate which can be find using SLAM
2. Obstacle Awareness which can be done using a static map and for live object detection when navigating is done with the help of Live Lidar data. So we need both the things that's why both these information combined in one and called cost map. In which where high cost area means these area has to avoid and low cost area means there is a free space to navigate through.

Navigation also requires much more information other than these, such as robot size, hardware specifications and dynamics of the robot etc.

To implement AMCL, components from the ROS nav2 stack are utilized. For a static map, the ROS2 AMCL module provides a node for localization. This node is connected to the Transform Features (TF) data stream., the static map that was previously constructed, and the 2D laser scan data from the robot. The AMCL node publishes the robot's pose and its estimated position with respect to the map. "Fig. 10" shows the rqt_graph for the AMCL. Data on the challenges in the virtual environment is presented in two cost maps. Across the map, global path planning is carried out by creating an extended travel itinerary using the global cost map. To design local routes and avoid obstacles, using the local cost map. The cost maps, both local and global, are displayed using Rviz. At this point, the robot can move on its own.

Using the 2D navigation objective in Rviz, a destination goal and direction are provided for the map. The robot plans its route to the target place and tells the robot controller to move at a certain speed. The robot's route from its starting place to the designated destination is seen in "Fig.10".

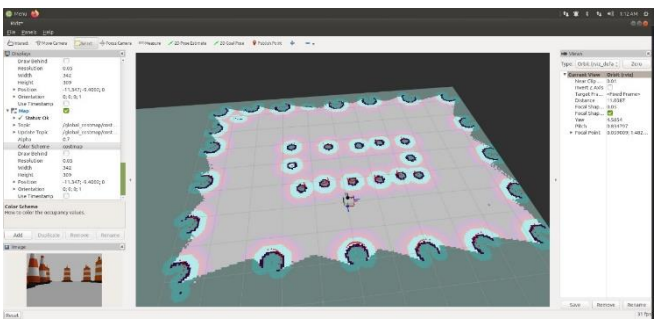

Fig 10.1 Initial Goal Pose

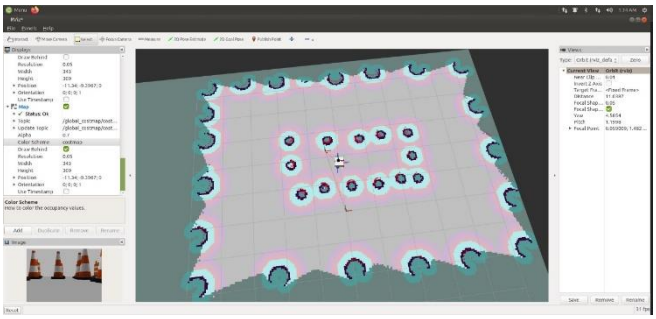

Fig 10.2 Center Goal Pose

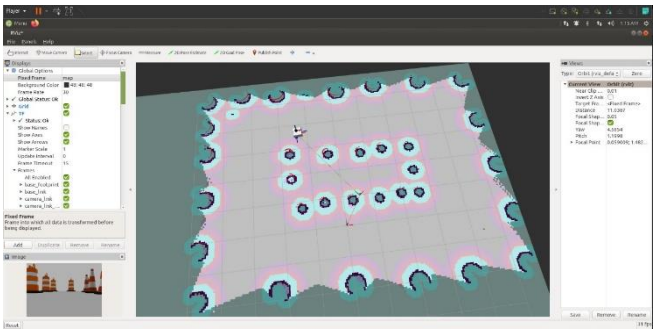

Fig 10.3 Top Goal Pose

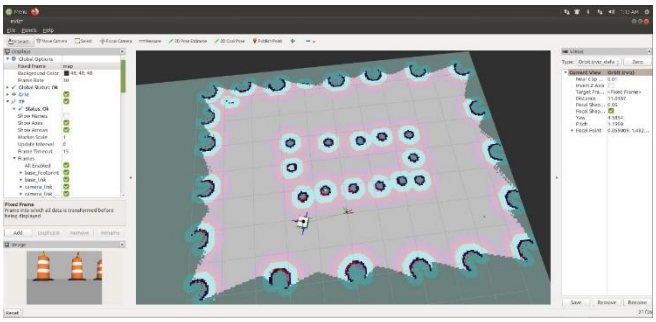

Fig 10.4 Bottom Goal Pose

Conclusion

In this study, we have successfully evaluated and implemented ROS2-based autonomous robot navigation. The preceding section covered the hardware and software implementation and logic. In order to accomplish this, we designed a custom robot that moves effortlessly through an agricultural field. Then, using that design, we emulated the movement of the robot in the Gazebo environment. We then implemented SLAM using the Grid Slam method, and with the help of the SLAM toolbox plugin, we were able to successfully produce and save a static map on the local system for use in the robot's navigation.

Following the creation of the SLAM, we were able to successfully navigate the environment using the ROS2 Nav2 Stack. This uses the map server, which makes use of the static map produced by the SLAM, along with AMCL and real-time 2D lidar data for real-time object avoidance. The map server then plans the path based on the objects that occur along the path to the goal position.

As of right now, our suggested technique can accomplish a respectable level of navigational precision. However, further work in the future will require even greater precision due to fully implemented hardware and calibrated maps, as well as more detailed descriptions of the robot.

- The desired destination (goal pose) for [5] the robot is defined. This tells the navigation system where the robot needs to go. The goal pose includes the robot's desired location and orientation at the destination.

DECLARATIONS

Conflict of Interest : The authors declare no competing interests.

Data Availability : No, we do not have any research data outside this manuscript.

Funding : The research has no funding associated with it

REFERENCES

[1] B. N. a. I. N. S. G. Eason, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," *Phil. Trans. Roy. Soc. London,* vol. vol. A247, no. (references), pp. 529-551, April 1955.

[2] J. C. Maxwell, "A Treatise on Electricity and Magnetism," *Oxford: Clarendon,* vol. vol. 2, pp. 68-73, 1892.

[3] I. S. J. a. C. P. Bean, "Fine particles, thin films and exchange anisotropy in Magnetism," *G. T. Rado and H. Suhl, Eds. New York,* no. Academic, pp. 271-350, 1963.

[4] K. Elissa, "Title of Paper if Known," *Unpublished.*

[5] H. I. M. A. O. a. K. S. M. Sahari, " "Indoor mapping using kinect and ROS,"," in *nternational Symposium on Agents, Multi-Agent Systems and Robotics (ISAMSR),*, Putrajaya, Malaysia, 2015.

[6] R. K. &. C. R. &. S. S. &. R. A. Megalingam, "ROS based Autonomous Indoor Navigation Simulation Using SLAM Algorithm.," 2019.

[7] D. M. Turnage, ""Simulation results for localization and mapping algorithms,"," in *2016 Winter Simulation Conference (WSC)*, Washington, DC, USA, 2016.