Revolutionizing Industrial Robotics: Smart Localizer's Autonomous Navigation with ROS2-Based SLAM Technologies

1st Srinivasa Rao Ch.
Department of Computer Science and Engineering
Lakireddy Bali Reddy College of Engineering (Autonomous)
Mylavaram, India
chsrao@lbrce.ac.in, 0000-0002-7351-7540

2nd Srikanth Kandi
Department of Computer Science and Engineering
Lakireddy Bali Reddy College of Engineering (Autonomous)
Mylavaram, India
srikanthkandi444@gmail.com, 0009-0001-8566-023X

3rd Joshith Sai Ram Sunkara
Department of Computer Science and Engineering
Lakireddy Bali Reddy College of Engineering (Autonomous)
Mylavaram, India
sjoshith04@gmail.com, 0009-0002-4485-6321

4th Jaya Surya Vemuri
Department of Computer Science and Engineering
Lakireddy Bali Reddy College of Engineering (Autonomous)
Mylavaram, India
suryarajs007@gmail.com, 0009-0002-6329-6489

***Abstract*—Robots have changed industrial operations over the last decade, reducing reliance on human labour. Our project introduces the "Smart Localizer," an autonomous robot that uses advanced Simultaneous Localization and Mapping (SLAM) techniques, which is especially useful in small regions where GPS fails. Using the Nav2 Stack, Adaptive Monte Carlo Localization (AMCL), and SLAM in ROS2, the robot combines 2D LiDAR and SLAM techniques to overcome GPS constraints in limited situations. This advancement makes precise mapping and localization possible. The Smart Localizer pioneers autonomous navigation, relieving it of human interaction and increasing efficiency and safety in industrial applications. Our initiative contributes to the growth of efficient and autonomous industrial robots by solving the constraints of limited spaces with cutting-edge technologies.**

***Index Terms*—Global Positioning System (GPS), SLAM, AMCL, ROS2, 2D LiDAR, Precision Mapping**

I. Introduction

The integration of robotics has resulted in a significant shift in the industrial operations environment during the last ten years, with a notable decrease in the dependence on human labor. In the quest to improve productivity and security in industrial settings, our work presents a novel autonomous robot called the "Smart Localizer." This robotic breakthrough makes use of sophisticated Simultaneous Localization and Mapping (SLAM) techniques that are designed to overcome difficulties in small areas where the Global Positioning System (GPS) is deemed insufficient [8].

Accurate mapping and localization in small places has long been a problem in the field of industrial robotics. In confined areas, traditional navigation systems—GPS in particular—have drawbacks. By utilizing cutting-edge technologies like the Navigation2 Stack, SLAM and Adaptive Monte Carlo Localization (AMCL) within the Robot Operating System 2 (ROS2) architecture [10], the Smart Localizer is made to overcome these limitations.

The Smart Localizer's primary technological basis is its skillful application of SLAM algorithms and 2D Li- DAR sensors [4]. The robot can now move autonomously in confined spaces—a capability GPS technology is illsuited for—thanks to this intentional integration. A strong foundation for the Smart Localizer is supplied by the Navigation2 Stack, which makes it easier to make wise decisions and navigate through challenging situations. Robust selflocalization, which is essential for maneuvering in limited areas, is further improved by the Adaptive Monte Carlo Localization (AMCL) technique.

The Smart Localizer's innovation is especially noteworthy since it helps get over GPS's restrictions in constrained environments. GPS, which is mostly intended for outside use, finds it difficult to provide precise and trustworthy data in the constrained spaces of industrial environments. The Smart Localizer can precisely localize itself even in difficult areas by incorporating SLAM technology to build detailed, real-time maps of its surroundings.

Our initiative's focus on autonomous navigation is one of its unique aspects. Unlike traditional robot systems, which frequently need human assistance for regular tasks, the Smart Localizer is a step forward. The Smart Localizer increases operational efficiency and promotes a safer workplace by doing away with the need for human interaction. The combination of AMCL, SLAM, and Navigation2 Stack technologies in ROS2 enables the Smart Localizer to travel across dynamic industrial settings intelligently, avoiding obstacles [9] and finding the best route.

In addition to addressing the current issues in industrial robotics, this research advances the development of autonomous systems more broadly. The Smart Localizer is an

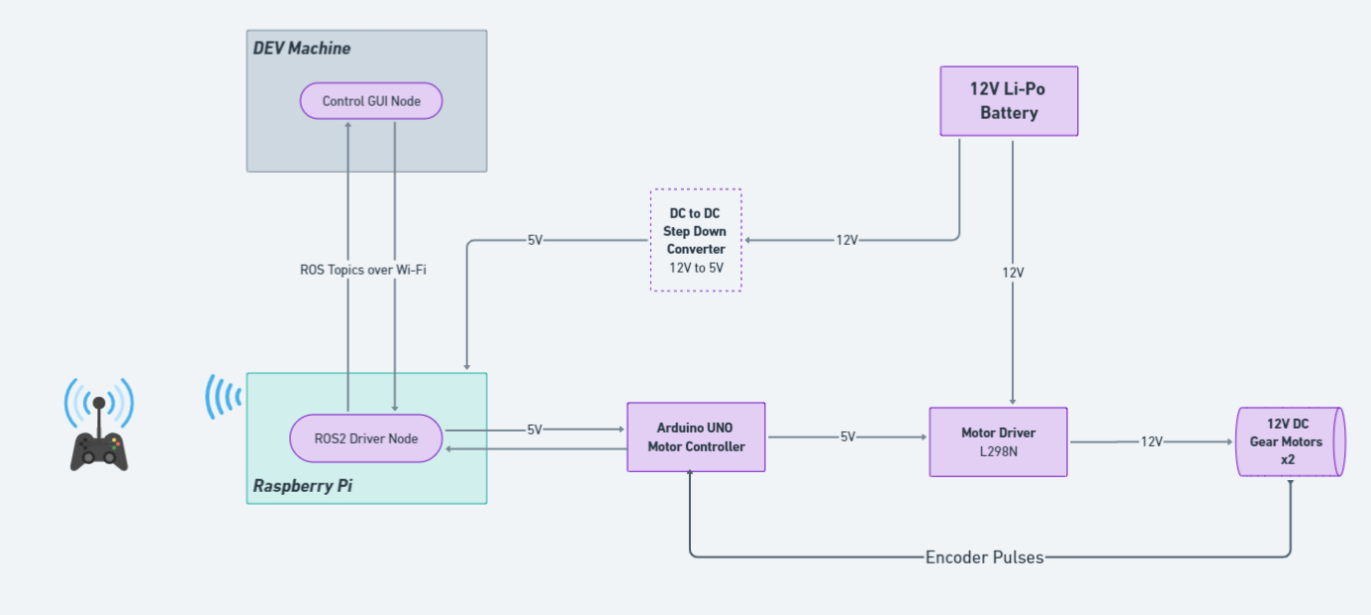


Fig. 1: Overall Structure of Smart Localizer

example of how cutting-edge technologies can be integrated to solve useful issues in real-world situations. Through creative solutions to space limitations, our project contributes to the direction of industrial automation, propelling the development of effective and self-governing robotic applications.

In conclusion, our endeavour takes place in the context of a decade that has revolutionized industrial operations. The Smart Localizer's launch demonstrates how sophisticated robots may be used to solve particular problems pertaining to small areas. The Smart Localizer ushers in a new era of autonomous navigation by strategically integrating state-of-the- art technology and SLAM techniques into ROS2. This integration promises better efficiency and safety in industrial applications. This study not only offers technological progress but also establishes the foundation for further developments in the field of effective and self-governing industrial robots.

II. Materials and Methods

As shown in Fig. 1, the components of the Smart Localizer are Raspberry Pi, Developer Machine, Arduino Uno Board, Power Bank more than 10,000 mAh, DC to DC Booster Converter from 5V to 12V, L298N Motor Driver, 12V DC Gear Motors with Encoders builtin.

A. Raspberry Pi

This is the brain of the Smart Localizer which accesses the control signal from the Developer Machine processes it and passes it as the appropriate signal for Arduino and vice-versa. For the reproduction of this project, any kind of Raspberry Pi Model will work as it supports 4 GB or more RAM. For our use case, we had chosen the Raspberry Pi 4 Mobel B with 4 GB RAM which is sufficient to run all the ROS2 nodes as we discussed further, and it was equipped with the Ubuntu Server 20.04 and ROS2 Foxy Fitrozy was installed.

B. Developer Machine

This can be any Laptop/Desktop equipped with Ubuntu or any other Debian-based distro, and the supported Robot Operating System (ROS2) was installed. For our use case, we had chosen the Ubuntu Desktop 20.04 machine was installed with ROS2 Foxy Fitrozy as choosing the different versions of ROS2 on Dev Machine and Smart Localizer can lead to errors at runtime.

C. Arduino as Motor Controller

The Arduino Uno Board was connected to the Raspberry Pi [II-A] via Serial over USB which helps both for the power supply of 5V as well as the management of control signals received. The Arduino piggybacks the $+5V$ and GND to the L298N Motor Driver and connects the IN_1, IN_2, IN_3, IN_4 pins of the Motor Driver to the Digital Pins D_6, D_{10}, D_9 and D_5 using Male to Female Jumper wires.

D. L298N Motor Driver

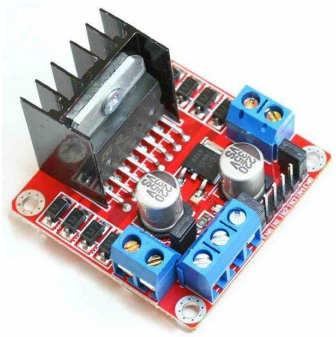


Fig. 2: Top View of L298N Motor Driver

This kind of motor driver is mostly used for Differential Drive Robots [7] as this motor driver at most supports up to 2 DC Motors, and controls their speed using the Proportional Integral Derivative (PID) control [5]. This intake the 5VPWM signal from the Arduino Uno [II-C] and converts to 12VPWM signal by combining the +12V DC from DC to DC Booster [II-E]. The Output A and Output B terminals of this motor driver are connected to Motor $+$ and Motor $-$ and altering the power flow in these terminals can make the motors rotate clockwise/anti-clockwise.

E. DC to DC Step Up Booster

The purpose of this DC to DC Step Up Power Booster is that the L298N Motor Driver requires a 12V DC power supply, as we decided to use the power bank as our main power source, it only outputs the 5V DC. So this component can help the Smart Localizer to work with a single power source rather than multiple power sources for 12V and 5V DC. Converting from $+5\text{V}$ & -5V to $+12\text{V}$ & -12V DC.

F. 12V DC Gear Motors with Encoders

Encoder motors are a special kind of motors that are used for the feedback given by the encoder pins provided within the motors. This feedback generally contains information related to the motor speed, number of rotations by the magnetic disk and so on.

Fig. 3: 12V 75RPM Metal Gear Motor with D type shaft

As shown in Fig 3, this kind of motor generally has 6 pins and they are

1) Motor +12V
2) Motor -12V
3) Encoder + (3.3V/5V)
4) Encoder -
5) Encoder Phase A
6) Encoder Phase B

Note: The order of the pins may vary on different motor manufacturers

The Left Motor's Encoder A, B and Right Motor's Encoder A, B output pins should be connected to the Arduino Uno board pins as D_2, D_3, A_4, and A_5 for inputs.

This feedback control is very crucial for Smart Localizer, as once it starts to move autonomously based on 2D Nav Goal Pose, the Raspberry Pi uses these encoder values to make the precise estimation of how many revolutions it requires for the motors to reach the destination accurately. This kind of control is called Closed-Loop control [6].

III. Communication using ROS2 Control

The `ros2_control`[1] is a framework provided by the ROS2 Foxy, this framework is used to control the robots in both simulation and real with minimal code change.

For our project, we decided to use the gamepad for the teleoperation of the robot, so the `ros2_control` [10] acts as a bridge between the robot actuators and the gamepad processing the necessary twist signals from the gamepad to appropriate command velocity for the robot.

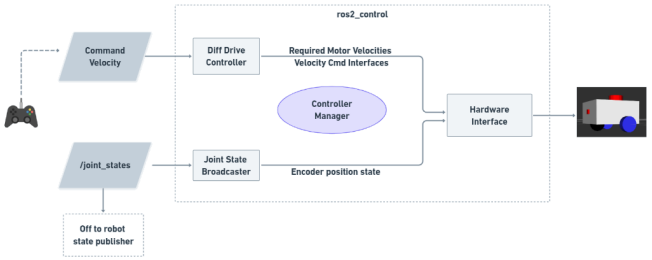


Fig. 4: Overview of ROS2 Control

As shown in Fig 4, the gamepad will send signals to the ROS topic Twist/TwistStamped and the Command Velocity node will subscribe to that topic constantly listening to changes in this topic, as the movement of our robot is constrained to moving forward/backwards in X-axis (i.e., `linear.x`) and rotating around Z-axis (i.e., `angular.z`).

The Controller Manager as shown in Fig 4, was provided by the `ros2_control` framework. It is often represented as (`ros2_control_node`) and the Diff Drive Controller is also available as a plugin in `ros2_control` and often represented as (`diff_drive_controller`).

The Hardware Interface requires a plugin which needs to be provided by ourselves, we found it under open-source.[2]

The Diff Drive Controller will convert the Command Velocity to the required motor velocities and the controller manager of `ros2_control` will help pass the velocity signal to the Hardware Interface by converting them to Velocity Command Interfaces.

As discussed in section, these encoder motors will return the feedback to the Controller Manager. The Joint State Broadcaster is another controller in `ros2_control` which listens to changes in the motor's encoder values and publishes them to `/joint_states` ROS topic.

Changes in `/joint_states` topic will trigger the change in the position of Robot Transform positions in Rviz2.

IV. Gamepad for Teleoperation

Teleoperation is the ability to remotely control the speed/velocity of the robot by a Human operator. For our use case, we decided to choose the gamepad with a USB dongle for teleoperation.

[1]https://control.ros.org/foxy/

[2]https://github.com/joshnewans/diffdrive_arduino

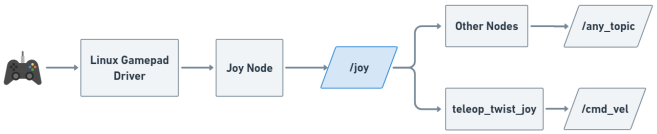


Fig. 5: Flow of Twist messages from Gamepad

The Linux Gamepad Driver installed within Raspberry Pi [II-A], which listens to changes in signals of the gamepad and passes them to ROS2 Joy Node which publishes the data to `/joy` topic.

The changes in axes of the thumb sticks trigger the `teleop_twist_joy` ROS2 package which will be remapped to the required Command Velocity in `/cmd_vel` ROS topic.

V. Robot Structure in Virtualization

To create the Robot structure within the virtualization, ROS2 provides support for Unified Robot Description Format (URDF) which contains a similar syntax to Extensible Markup Language (XML). This URDF can be used to create the 3D model of the Robot and also contains information related to mass, motor position, joints, lidar and caster wheel positions.

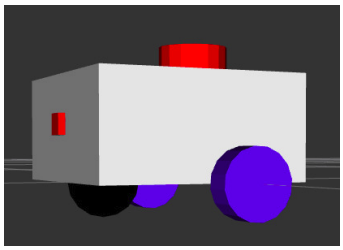


Fig. 6: 3D Model of Smart Localizer

All the code related to this URDF and driving the robot in the simulation are available open-source.[3]

To drive the robot virtually, ROS2 provides a package called `teleop_twist_keyboard`.[4] by which we can drive the robot within the simulation tools like Gazebo and Rviz2 using the keyboard keys.

But as we discussed in Gamepad for Teleoperataion section, this keyboard way of controlling the robot in both simulation and real is hard and sometimes this can also trigger the movement of the robot in accidental key presses.

After cloning the `my_bot`[3] into the ROS2 workspace, and build the workspace using Colcon build tool. We have decided to configure the gamepad buttons `L1` for the enable button and `R1` for the enable turbo button or to move the robot in a faster way.

But to configure our gamepad buttons within ROS, there is a package available open-source[5] called a `joy_tester`. Clone this `joy_tester` repository into the same workspace as `my_bot` and again build the workspace using Colcon build tool.

Open the ROS workspace parent directory in the terminal, source the workspace and run the below command to create the joy node and publish the data to `/joy` topic.

```
ros2 run joy joy_node
```

Make sure to connect your gamepad to the computer, before running the above command. Now open the new terminal tab of the ROS workspace, source it and run the below command to execute the `joy_tester` package.

```
ros2 run joy_tester test_joy
```

This command will bring up the popup window, showing all the assigned numbers for gamepad buttons with the values for axis of the thumbsticks. By triggering the different buttons of gamepad we can get the ROS assigned numbers for `L1` and `R1` buttons of our gamepad and we can modify the numbers for these buttons under `my_bot/config/joystick.yaml` file with keys `enable_button` and `enable_turbo_button`.

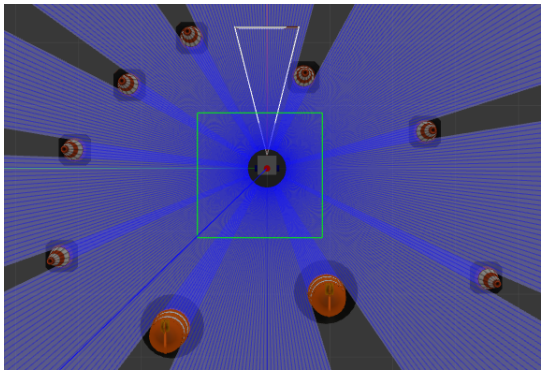


Fig. 7: Spawning robot in Gazebo world

To create a 3D world for the robot to move within, ROS provides a simulator called Gazebo.[6] By the help of this simulator we can load an already created virtual world/create a world by ourselves with using the 3D model figures provided online and save it to `.world` file.

The Fig 7, shows the initial spawn of the robot in the Gazebo world listed under the `my_bot/worlds/obstacles.world` file. The blue lines shown in this figure are the LiDAR rays placed on top of the Robot URDF as shown in Fig 6.

This `obstacles.world` file contains some construction cones and construction barrels placed randomly.

For the real assembly of our robot we used the LD19 LiDAR of the D300 LiDAR Kit provided by LDROBOT[7] has specifications like 0.03-12m measuring range, 5-13Hz Light Frequency with 360° field of view.

[3]https://github.com/srikanth-kandi/my_bot
[4]https://index.ros.org/r/teleop_twist_keyboard/\#foxy
[5]https://github.com/joshnewans/joy_tester
[6]https://gazebosim.org/docs
[7]https://www.ldrobot.com/ProductDetails?sensor_name=D300+Kit

The LDROBOT Sensor Team provides the support for ROS2 integration of LD19 LiDAR by making the SDK available open-source at `ldlidar_stl_ros2`[8].

All the required files for the same configuration of the LD19 LiDAR sensor were made available within the `my_bot`[3] repository.

To drive the spawned robot within the Gazebo world, the configured gamepad's `L1` button needs to be pressed initially. This switch was configured as the dead man's switch.

After that gamepad control will be activated by the `ros2_control` and to move the robot we need to hold either `L1`/`R1` button and move the left joystick in the gamepad for movement of the robot.

There is a `launch_sim.launch.py` file in `my_bot/launch/` directory, this ROS2 launch file contains the necessary nodes like rsp, diff drive spawner, joint broad spawner, gazebo, joystick and twist multiplexer which are required to work within the simulation environments like Gazebo and Rviz2.

VI. Map creation using SLAM Toolbox

SLAM Toolbox [8] is an open-source[9] package for Simultaneous Localization and Mapping and it was integrated into all versions of ROS2.

The SLAM Toolbox provides three modes of operations Synchronous, Asynchronous and Pure Localization. For our project, we have chosen Online Asynchronous mode for the creation of a map using LiDAR data.

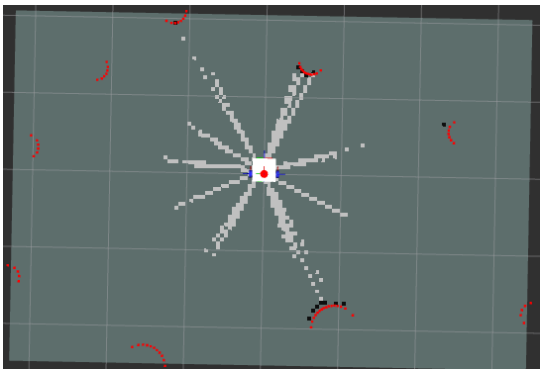


Fig. 8: Initial map created before traversing in Gazebo world

Online means to create the map from the live LiDAR data rather than stored on previous sessions and Asynchronous means to parse only the latest LiDAR data received from the sensor.

The Fig 8, shows the initial creation of the map after the Online Asynchronous SLAM was initialized for Smart Localizer within Gazebo world, all the settings required for map creation using SLAM were configured to `my_bot/config/slam_bot.rviz`

The Fig 9, shows the final map created after the robot has moved within the Gazebo world. Each white pixel represents the 5cm precision within a real-world map and obstacles surrounded by black pixels need to be avoided in navigation mode. SLAM Toolbox provides a plugin for Rviz2 which contains options to save the map to `.pgm` and `.yaml` file extensions. Saving the created maps helps to load maps for future map expansion or to feed those maps to the Navigation2 Map Server.

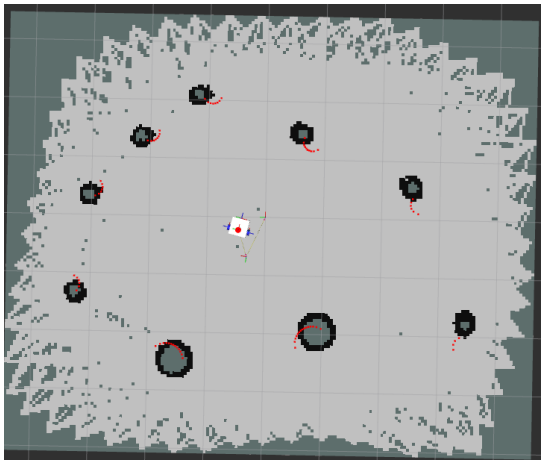


Fig. 9: Completed map after traversing in Gazebo world

VII. Autonomous Navigation using Nav2 Stack

The Nav2 stack[10] is a bundle of packages provided by Open Navigation LLC for the ROS2. It provides a Navigation System [9] for robots that are easily integrated into Simultaneous Localization and Mapping.

As shown in Fig 11, the navigation stack estimates the robot position based on SLAM, and obstacle detection and avoidance based on the Live LiDAR data along with referencing the static map created during SLAM.

Nav2 parameters file `nav2_params.yaml` and launch files such as `navigation_launch.py` and `localization_launch.py` was integrated to `my_bot`.

After the activation of the Nav2 stack, it creates a Global Costmap representing 3 shades of colours around a detected obstacle shown in Fig 10. The outer portion of an obstacle is Low cost, the middle portion is represented as Medium cost and the inner portion of the obstacle is represented as High cost.

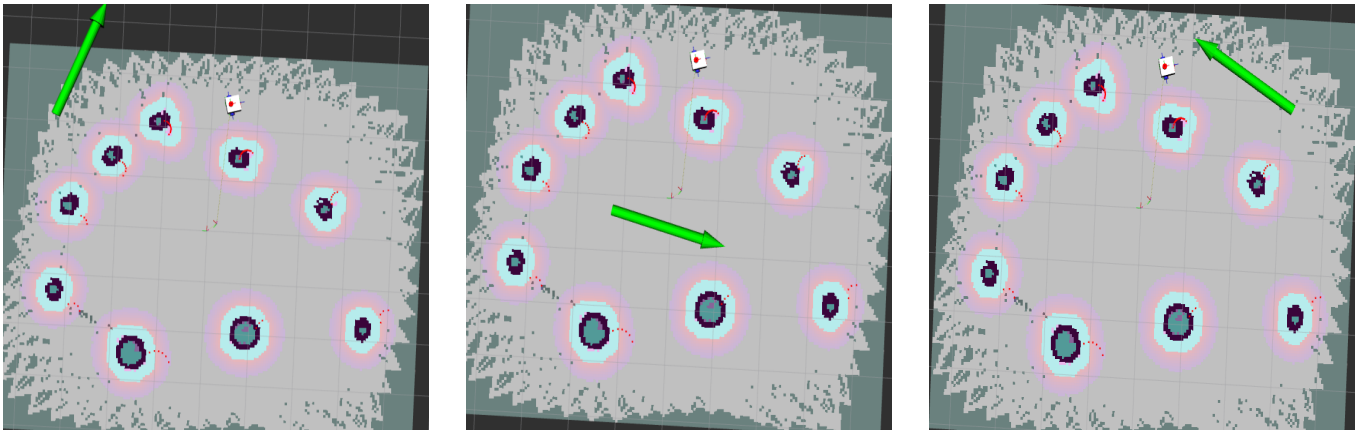
Nav2 stack provides a plugin for Rviz2, we can add a Navigation 2 panel with a Waypoint mode option after activating it, we need to specify multiple waypoints or destinations required for the robot to reach. As shown in Figures 10a, 10b and 10c we have specified 3 random waypoints for the Smart Localizer to reach within the map.

After specifying these waypoints, once we clicked the Start Navigation button on the Navigation 2 panel of Rviz2, the Figures 10d, 10e and 10f represents the path Smart Localizer had moved to those waypoints.

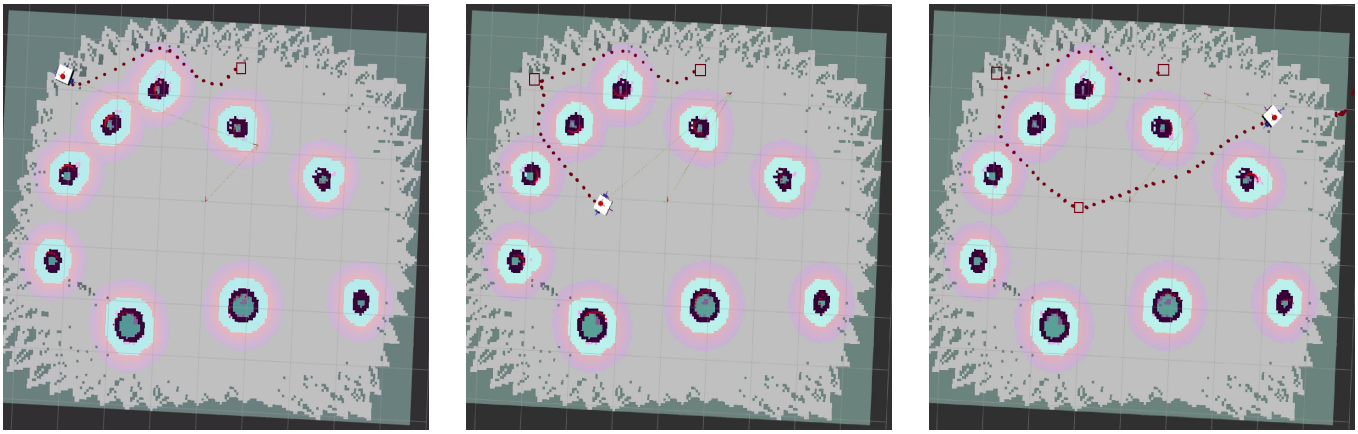
[8]https://github.com/ldrobotSensorTeam/ldlidar_stl_ros2

[9]https://github.com/SteveMacenski/slam_toolbox

[10]https://navigation.ros.org/



(a) Specifying the Nav2 Goal Pose 1 (b) Specifying the Nav2 Goal Pose 2 (c) Specifying the Nav2 Goal Pose 3



(d) Robot reached Goal Pose 1 destination (e) Robot reached Goal Pose 2 destination (f) Robot reached Goal Pose 3 destination

Fig. 10: Destination Path of Smart Localizer in specifying multiple Navigation 2 Goal Poses on Waypoint mode

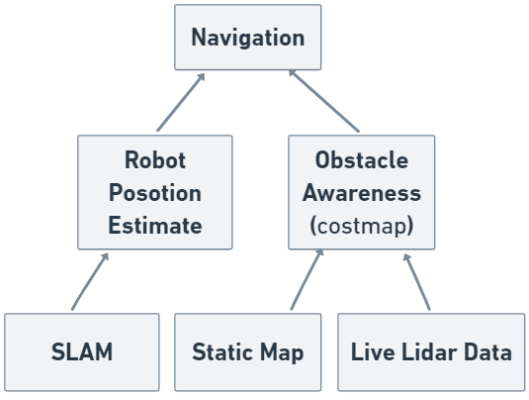


Fig. 11: Flow of Navigation 2

During this navigation, if the Smart Localizer had chosen the wrong way for the destination we can retake the control to gamepad control using the `twist_mux`[11] package provided by ROS2, The required parameters for the file are mentioned under the `my_bot/config/twist_mux.yaml`. This `twist_mux` package is used to multiplex the control signal received from the gamepad as a higher priority and once we stop the gamepad signal, the control will be again taken back by Nav2 to reach the destination autonomously.

[11]https://wiki.ros.org/twist_mux

VIII. Smart Localizer built using Hardware

The entire hardware circuit was set on to the Electric switchboard with a square shape, where the 2 encoder motors were placed at the back side of the robot and a supporting caster wheel was positioned in front of the robot with the middle axis to the two wheels at the back. So there will be a wooden plate that covers the top of the robot and had a LiDAR sensor placed on it supported with screws tightened into the wood. This wooden plate was supported by a tape holding the both Electric switchboard and the plate. To control this whole circuit there will be a switch at the side as shown in Fig. 12. The power supply will be entirely managed by the 12V DC LiPo battery which will be step down to 5V to power the Raspberry Pi, Arduino and Motor Driver and the 12V will be passed to the motor driver as well as the motor's power supply.

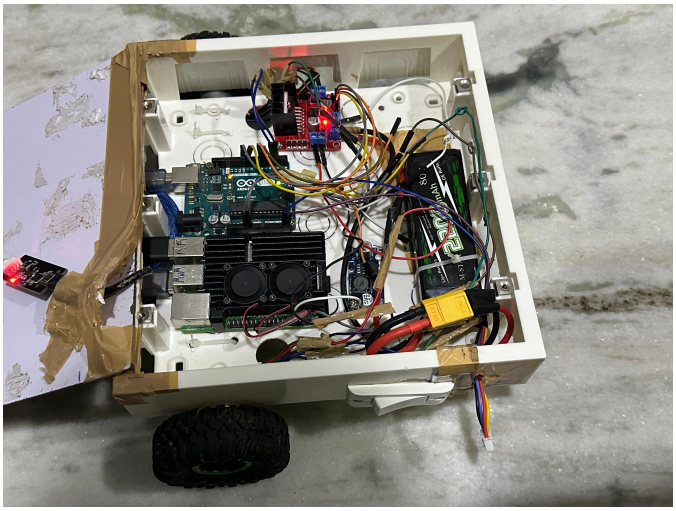


Fig. 12: Top view of Smart Localizer while opened

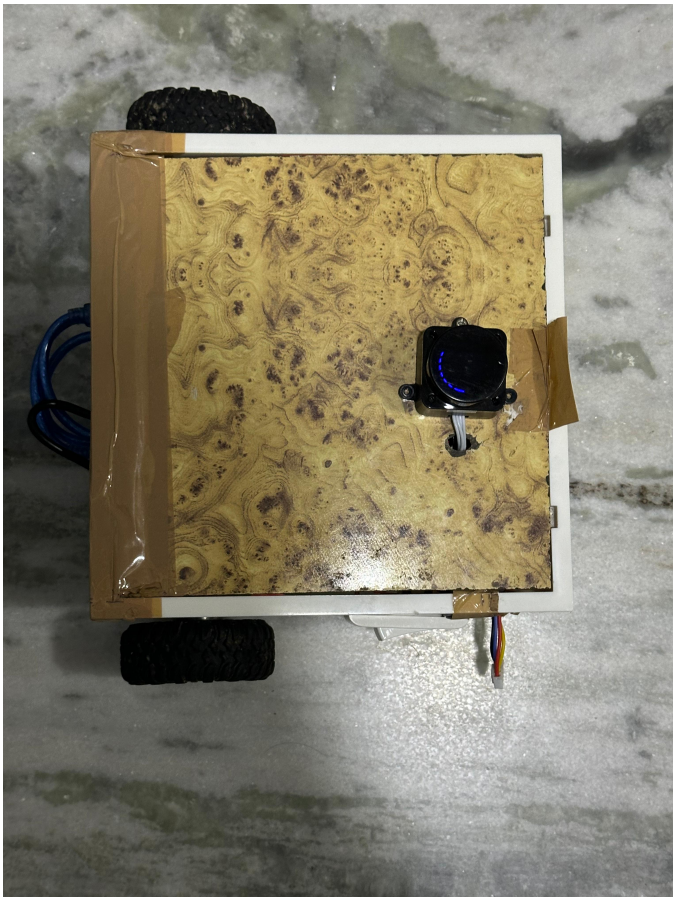


Fig. 13: View of Smart Localizer while closed

IX. Conclusion

In conclusion, our project greatly contributes to Industrial Automation addressing the challenges of confined spaces where traditional Global Positioning Systems (GPS) fall short. The ”Smart Localizer” showcases the successful integration of Simultaneous Localization and Mapping (SLAM), Navigation 2 stack and Adaptive Monte Carlo Localization tailored for Robot Operating System 2 based on 2D LiDAR. This not only enables precise mapping using SLAM but also makes the robot move autonomously by detecting and avoiding obstacles on the fly. This not only enhances the efficiency of Industries but also enables the safety standards in Industrial automation.

References

[1] Pan, Shuang, Zihui Xie, and Yulian Jiang. ”Sweeping robot based on laser SLAM.” Procedia Computer Science 199 (2022): 1205-1212.

[2] Huang, Leyao. ”Review on LiDAR-based SLAM techniques.” 2021 International Conference on Signal Processing and Machine Learning (CONF-SPML). IEEE, 2021.

[3] Xiaoyu, Wang, et al. ”On adaptive monte carlo localization algorithm for the mobile robot based on ROS.” 2018 37th Chinese Control Conference (CCC). IEEE, 2018.

[4] Song, Kai-Tai, et al. ”Navigation control design of a mobile robot by integrating obstacle avoidance and LiDAR SLAM.” 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC). IEEE, 2018.

[5] Ariyansyah, Qolil, and Alfian Ma’arif. ”DC Motor Speed Control with Proportional Integral Derivative (PID) Control on the Prototype of a Mini-Submarine.” Journal of Fuzzy Systems and Control 1.1 (2023): 18-24.

[6] Naveen, V., and T. B. Isha. ”A low cost speed estimation technique for closed loop control of BLDC motor drive.” 2017 International Conference on Circuit, Power and Computing Technologies (ICCPCT). IEEE, 2017.

[7] Dignadice, Sherwin John, et al. ”Application of Simultaneous Localization and Mapping in the Development of an Autonomous Robot.” 2022 8th International Conference on Control, Automation and Robotics (ICCAR). IEEE, 2022.

[8] Macenski, Steve, and Ivona Jambrecic. ”SLAM Toolbox: SLAM for the dynamic world.” Journal of Open Source Software 6.61 (2021): 2783.

[9] Macenski, Steve, et al. ”The marathon 2: A navigation system.” 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2020.

[10] Erős, Endre, et al. ”A ROS2 based communication architecture for control in collaborative and intelligent automation systems.” Procedia Manufacturing 38 (2019): 349-357.