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

A

Project Report Submitted In partial fulfilment of the requirements for the award of the Degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE & ENGINEERING

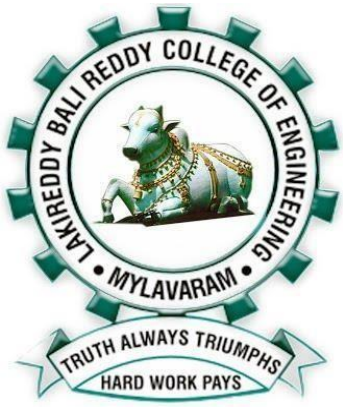
By

Kandi Srikanth	**20761A0589**
Sunkara Joshith Sai Ram	**20761A05C2**
Vemuri Jaya Surya	**21765A0512**

Under the esteemed guidance of

Ch. Srinivasa Rao

Assistant Professor



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

LAKIREDDY BALI REDDY COLLEGE OF ENGINEERING

(AUTONOMOUS)

Accredited by NAAC with 'A' Grade & NBA (Under Tier - I), ISO 9001:2015 Certified Institution

Approved by AICTE, New Delhi and Affiliated to JNTUK, Kakinada

L.B. REDDY NAGAR, MYLAVARAM, NTR DIST., A.P.-521 230.

2020-2024

LAKIREDDY BALI REDDY COLLEGE OF ENGINEERING (AUTONOMOUS)

Accredited by NAAC with 'A' Grade & NBA (Under Tier - I), ISO 9001:2015 Certified Institution

Approved by AICTE, New Delhi and Affiliated to JNTUK, Kakinada

L.B. REDDY NAGAR, MYLAVARAM, NTR DIST., A.P.-521 230.

Department of

COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the project entitled "**Revolutionizing Industrial Robotics: Smart Localizer's Autonomous Navigation with ROS2-Based SLAM Technologies**" is being submitted by

Kandi Srikanth	**20761A0589**
Sunkara Joshith Sai Ram	**20761A05C2**
Vemuri Jaya Surya	**21765A0512**

In partial fulfilment of the requirements for the award of degree of **B.Tech** in **Computer Science & Engineering** from **Jawaharlal Nehru Technological University Kakinada** is a record of bonafide work carried out by them at **Lakireddy Bali Reddy College of Engineering.** The results embodied in this Project report have not been submitted to any other University or Institute for the award of any degree or diploma.

PROJECT GUIDE
Ch. Srinivasa Rao

HEAD OF THE DEPARTMENT
Dr. D. Veeraiah

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We take great pleasure to express our deep sense of gratitude to our project guide **Ch. Srinivasa Rao,** Assistant Professor, for his valuable guidance during the course of our project work.

We would like to thank **Dr. D. Veeraiah**, Professor & Head of the Department of Computer Science & Engineering for his encouragement.

We would like to express our heart-felt thanks to **Dr. K. Appa Rao,** Principal, Lakireddy Bali Reddy College of Engineering for providing all the facilities for our project.

Our sincere gratitude to **Dr. B. Siva Rama Krishna**, Associate Professor, to his contribution and support throughout our project work. Our utmost thanks to all the faculty members and Non-Teaching Staff of the Department of Computer Science & Engineering for their support throughout our project work.

Our Family Members and Friends receive our deepest gratitude and love for their support throughout our academic year.

Kandi Srikanth	**20761A0589**
Sunkara Joshith Sai Ram	**20761A05C2**
Vemuri Jaya Surya	**21765A0512**

DECLARATION

We are here by declaring that the project entitled **"Revolutionizing Industrial Robotics: Smart Localizer's Autonomous Navigation with ROS2-Based SLAM Technologies"** work done by us. We certify that the work contained in the report is original and has been done by us under the guidance of our supervisor. The work has not been submitted to any other institute in preparing for any degree or diploma. We have followed the guidelines provided by the institute in preparing the report. We have confirmed to the norms and guidelines given in the Ethical Code of Conduct of the Institute. Whenever we have used materials (data, theoretical analysis, figures and text) from other sources, we have given due credit to them by citing them in the text of the report and giving their details in the references. Further, we have taken permission from the copyright's owner of the sources, whenever necessary**.**

Signatures of the students

Kandi Srikanth	**20761A0589**
Sunkara Joshith Sai Ram	**20761A05C2**
Vemuri Jaya Surya	**21765A0512**

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.

LIST OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

1. IoT - Internet of Things
2. DC - Direct Current
3. FMCW - Frequency Modulated Continuous Wave
4. HD - High Definition
5. LIDAR - Light Detection and Ranging
6. RADAR – Radio Detection and Ranging

1. INTRODUCTION

1.1 Overview of the project

The internet of things, or IoT, is a system of interrelated computing devices, mechanical and digital machines, objects, animals, or people that are provided with unique identifiers (UIDs) and the ability to transfer data over a network without requiring human-to-human or human-to computer interaction. A thing on the internet of things can be a person with a heart monitor implant, a farm animal with a biochip transponder, an automobile that has built-in sensors to alert the driver when tire pressure is low or any other natural or man-made object that can be assigned an Internet Protocol (IP) address and is able to transfer data over a network.

An IoT ecosystem consists of web-enabled smart devices that use embedded systems, such as processors, sensors and communication hardware, to collect, send and act on data they acquire from their environments. IoT devices share the sensor data they collect by connecting to an IoT gateway or other edge device where data is either sent to the cloud to be analysed or analysed locally. Sometimes, these devices communicate with other related devices and act on the information they get from one another. The devices do most of the work without human intervention, although people can interact with the devices -- for instance, to set them up, give them instructions or access the data. With the increase of technological advancements, everyone wants multitask-enabled products. In the past, numerous robotic cars were built to perform certain type of tasks. With this project, a wide scope of capacities is incorporated into a solitary model of a military automated vehicle. It is the robot which autonomously detects obstacles in front of it and does pick and place function.

IoT focuses on supporting services for pervasive sensing, monitoring, and tracking, while the robotic communities focus on production action, interaction and autonomous behaviour. A concept where the sensor data is coming from a range of sources is fused, processed with local and distributed intelligence, used to control, manipulate objects in the physical world is how the term "Internet of Robotic Things" was created. IoRT has three intelligent components: First, the robot can sense that it has embedded monitoring capabilities and can get sensor data from other sources. Second, it can analyse data from the event it monitors, which means there's edge computing involved. Edge computing is where the data is processed and analysed locally instead of cloud. It eliminates the need to transmit a wealth of data to the cloud. Third, because of the first two components, the robot can determine which action to take and then take that action. The robot can control or manipulate a physical object, and if it was designed to, it can move in the physical world.

The bigger idea for now is collaborations between machine / machine and between man / machine. These interactions could move toward predictive maintenance and entirely new services.

1.1 Feasibility Study

A feasibility study is an analysis that takes into account all pertinent project elements, such as technical, social, and economic ones. In this, we look at a number of scenarios in which the project could be finished with the hardware and software already in place. The project's economic viability determines whether it should be undertaken. The request and other studies were authorised as the first phase's outcome, and it was determined that the project would benefit the end user. This software's development and deployment save a significant amount of money and share important time.

• Technical feasibility; social feasibility; and economic feasibility

1.2.1 Economical Feasibility

The purpose of the study was to determine the potential financial impact of the device on the company. The amount of money going into the device's research and development is limited. The cost is warranted.

1.2.2 Technical Feasibility

This is accomplished to evaluate the system's technical viability or technical requirements. Any advanced system requires a modest demand for the technical resources available. The developed system must have reasonable requirements to enforce this machine. The task's technological needs are manageable.

1.2.3 Social Feasibility

The study aims to determine how much machine recognition can be accomplished with human assistance. This covers the method of teaching the user how to operate the system effectively. The device shouldn't put the user in danger. It is necessary to raise their level of self-belief so they can offer some helpful criticism, which is appreciated.

1.3 Scope

With the emergence of robots, modern life is getting more demanding and easier in every way. These robots offer a means of utilising the "Internet of Things" to create a "Connected Home." In our project, it serves as a subordinate based on the user's needs in an easy-to-understand manner by using a range of inputs. The trolly lessens humans' workload by distributing things from one person to another. Because of its intelligence, it can identify obstacles in its path on its own and change direction to reach the intended destination. The equipment reduces manual labour and does not require users to have any prior training. Without assistance from a human, the smart Localizer moves from one place to another inside a specific region.

2. LITERATURE SURVEY

The evolution of a self-driving car that used a Raspberry Pi was developed by Mohammad and Gurajashan Pannu in 2015. The techniques that they used were IR monitor and broadcaster, as well as the hadarsine algorithm. The research uses a Raspberry Pi as a processor chip to construct a monocular vision automated vehicle prototype. An HD camera and a sensor are used to give the car the information it needs about the outside world.

Ciprian-Florin and Duka completed a project in 2014 that involved automatically mapping a space using a robotic system. By using tools like movement modules, image acquisition features, and odometry. The ability to precisely position themselves while constantly creating a map of their surroundings is essential for an autonomous mobile robot navigating through an uncharted environment. The ROS-based obstacle detection robot using ultrasonic sensor and FMCW RADAR was published by Anusha T R, S. Selva Kumar and Swarnam Panday. Ultrasonic sensor and FMCW RADAR are the methods and techniques they used to develop this project. The main objective of their survey is to detect obstacles using a Robotic Operating System.

2.1 Existing System & Drawbacks

The voice command input on the smartphone is converted to text using the Android application. After then, this text based signal is delivered from the robot's Bluetooth module, which in turn transmits it to the microcontroller. Following signal processing, the microcontroller will move the robot appropriately. The robot may be contacted via a smartphone. Because distinct motors are utilized for the different sections of the robot, it is possible to control the movements of the robot's body, hands, and arm all at once. Disadvantages Only by using mobiles they can access the robot. It can cover up to some area.

Limitations:

- Only by using mobiles they can access the robot.
- It can lose connection in certain conditions.
- It has low bandwidth as compared to Wi-Fi.
- It allows only short-range communication between devices.

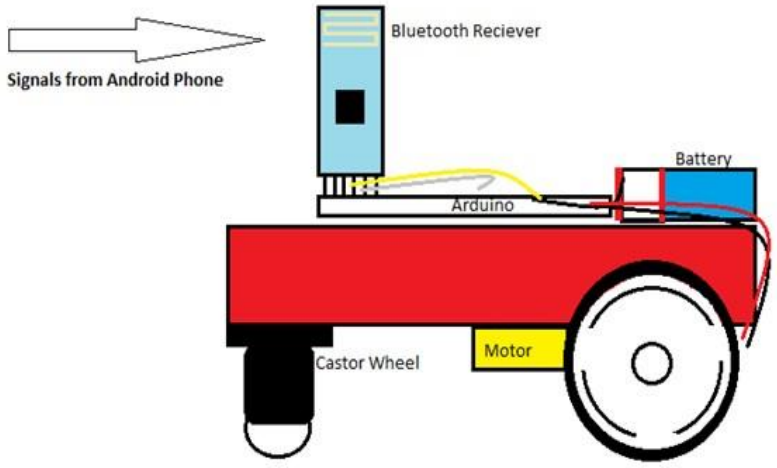


Fig.1: Simple image of Bluetooth controlled robot.

2.2 Proposed System

A mobile robot platform with a fixed two-wheel arrangement chassis that is built with the Raspberry Pi and Arduino Uno interfaces in mind has been proposed. The mobile platform satisfies several fundamental design criteria at this stage of development by offering a low-cost, hugely dependable, and expandable solution that is used in research as well as teaching (as an instructive platform for microcontrollers, electronics, automation, and robotics).

Motivation

- To decrease human efforts for small works.
- To increase productivity and reliability.
- Interests in robotics.
- Easy access without any android mobiles.

Steps for Working Procedure

Step 1: The connection between the host system and the robot is made when both are connected to same network or different network when port forwarding is used.

Step 2: The host system will logged in to the robot via SSH (Secure Shell)

Step 3: The required commands to activate the ROS nodes in the Raspberry Pi will be done by the SSH input, it activates the launch_robot.launch.py and ld19.launch.py

Step 4: After the successful activation of these two nodes, the Raspberry Pi will continuously read the data from LiDAR as well as the response from the Arduino UNO.

Step 5: Since the Arduino UNO holds all the GPIO pins such the L298N Motor Driver, Motor's encoder wires and acts as the power supply input of +5V and GND.

Step 6: On the host system we will connect a joystick receiver and activate all the other required ROS nodes for the successful configuration of the robot.

Step 7: To move the robot, hold the joystick's L1 button and use the left thumb stick to move forward, backward, left, and right.

Step 8: The robot captures the live LiDAR as sends it back to the host system, this data will be used by the SLAM to generate a map of the surroundings.

Step 9: Once the map is generated, we can activate the Nav2 Stack and specify the 2D Goal Pose, to make the robot move autonomously to the specified destination.

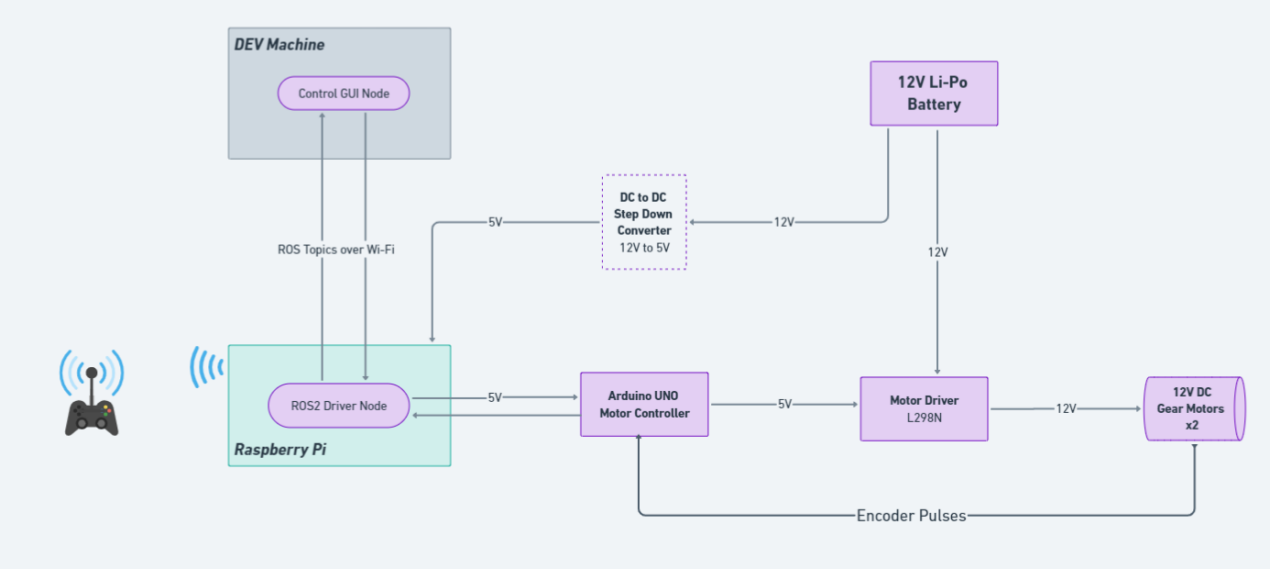


Fig.2: Flowchart of working of Smart Localizer

Fig.3: Smart Localizer

3. SYSTEM ANALYSIS

3.1 Overview of System Analysis

During the software development phase, various software development life cycle models are specified and designed. Another name for these models is "Software Development Process Models." To guarantee success at each stage of the software development process, each process model adheres to a set of phases specific to its kind. This model, a continuous software development approach, views software development as a cascading series of processes that include requirements analysis, design, implementation, testing (validation), integration, and maintenance.

There are some important ramifications to linear activity ordering. First, at the conclusion of each step, certain certification mechanisms must be used to indicate the end of one phase and the start of the next. In order to make sure that the stage's output is compatible with both its input—the output from the previous step—and the system's overall needs, some verification and validation normally go in this direction.

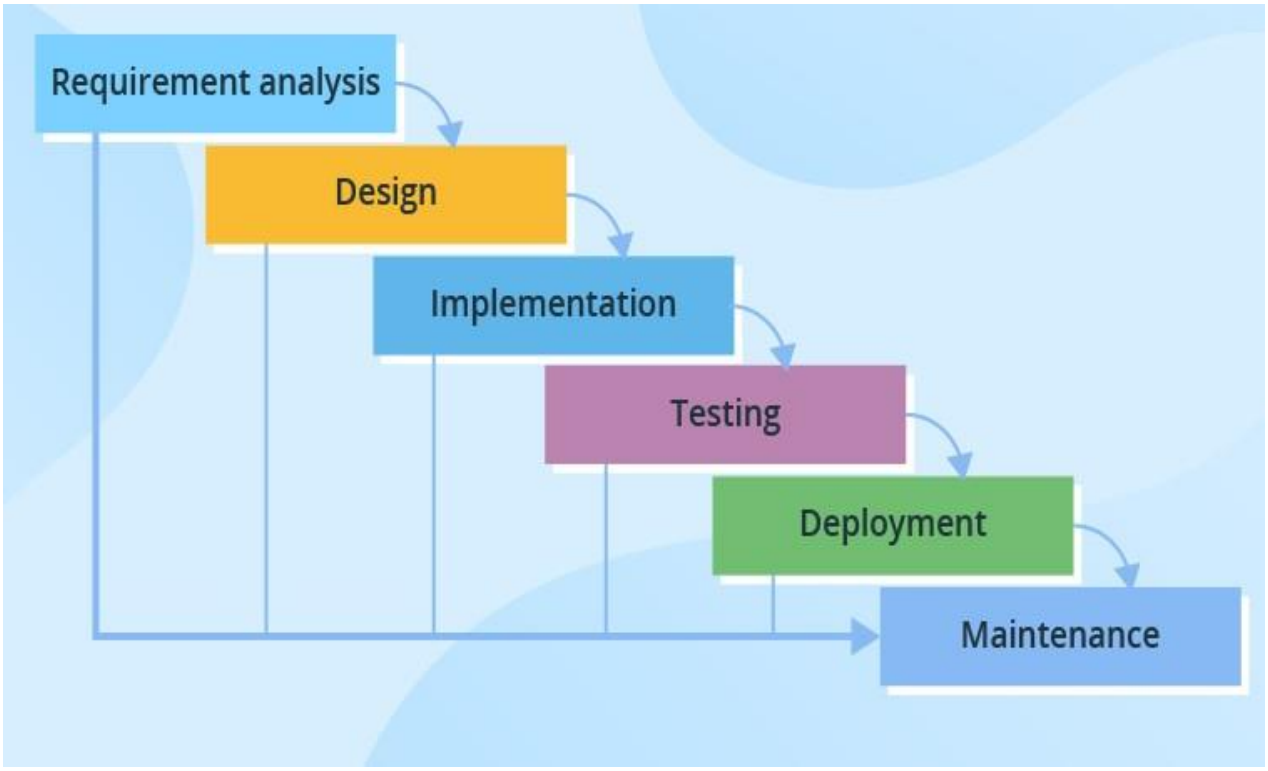


Fig.4: Waterfall model

3.1.1 Requisites Accumulating and Analysis

This is the first and most important stage of any project because it is an academic leave for gathering requirements. We followed IEEE journals and gathered many IEEE-related papers, culminating in a final paper designated by setting and important input. For the analysis stage, we gathered references from the paper and conducted a literature review of a few papers, gathering all the requirements for the project at this point.

3.1.2 System Design

In the Design Phase, the system is created to meet the needs that were determined in the earlier stages. A system design document that precisely outlines the system's design and can be utilised as a source of information for system development in a later phase is created using the requirements found in the requirements analysis phase.

3.1.3 Implementation

The majority of the coding for business logic is put into practice during the implementation phase, which is where we try to provide tangible results of the work done during the design stage. This is an important and pivotal element of the project.

3.1.4 Testing

Unit Testing

The developer completes every step of the project directly, and only here will we address all runtime errors through the refinement of the bug and extra developer-performed module.

Manual Testing

Since our project involves an academic leave, we are unable to conduct any automated testing; instead, we must conduct manual testing using trial and error techniques.

3.1.5 Deployment of System

We intend to implement the client system in the actual world upon project completion. We just installed it in our college lab with Windows OS and the required software while we were on academic leave.

3.1.6 Maintenance

The Maintenance of our Project is one time process only.

3.2 Software Used in the Project

The things you need to install the software:

- Arduino ide
- Visual Studio Code

3.3 SYSTEM REQUIREMENTS

3.3.1 Software Requirements

- Software: Arduino IDE, Visual Studio Code, ROS2, Rviz, Gazebo
- Version: ROS2 Foxy
- Operating System: Ubuntu 20.04

3.3.2 Hardware Requirements

- One Arduino Uno
- LD19 LiDAR Sensor
- Raspberry Pi version 4 with 4GB RAM and more
- L298N Motor driver module
- 12V DC Gear Motors with Encoders
- DC to DC Step Up Booster
- 12V Battery
- Robot wheels
- Robot chassis

4. SYSTEM DESIGN

4.1 Overview of System Design

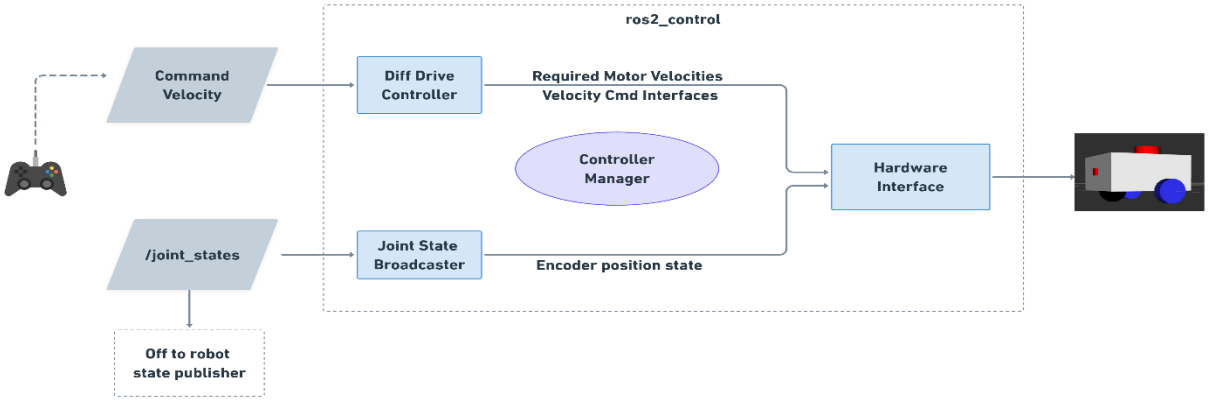


Fig.5: System design of Smart Localizer

System Design:

The project's goal was to build an Arduino-integrated robot that required control via an ROS2 host system. The project was effectively finished, much to everyone's satisfaction. Upgrades to the code are made possible. The app has produced health results and has been tested using real-time data. The code has, therefore, been shown to be hurried. The system's ease of use and implementation allowed it to accomplish its goals. Additional modules could be added later on if needed. The code is created using a uniform methodology. Every module in the system is tested with all the necessary data to ensure that everything functions as intended.

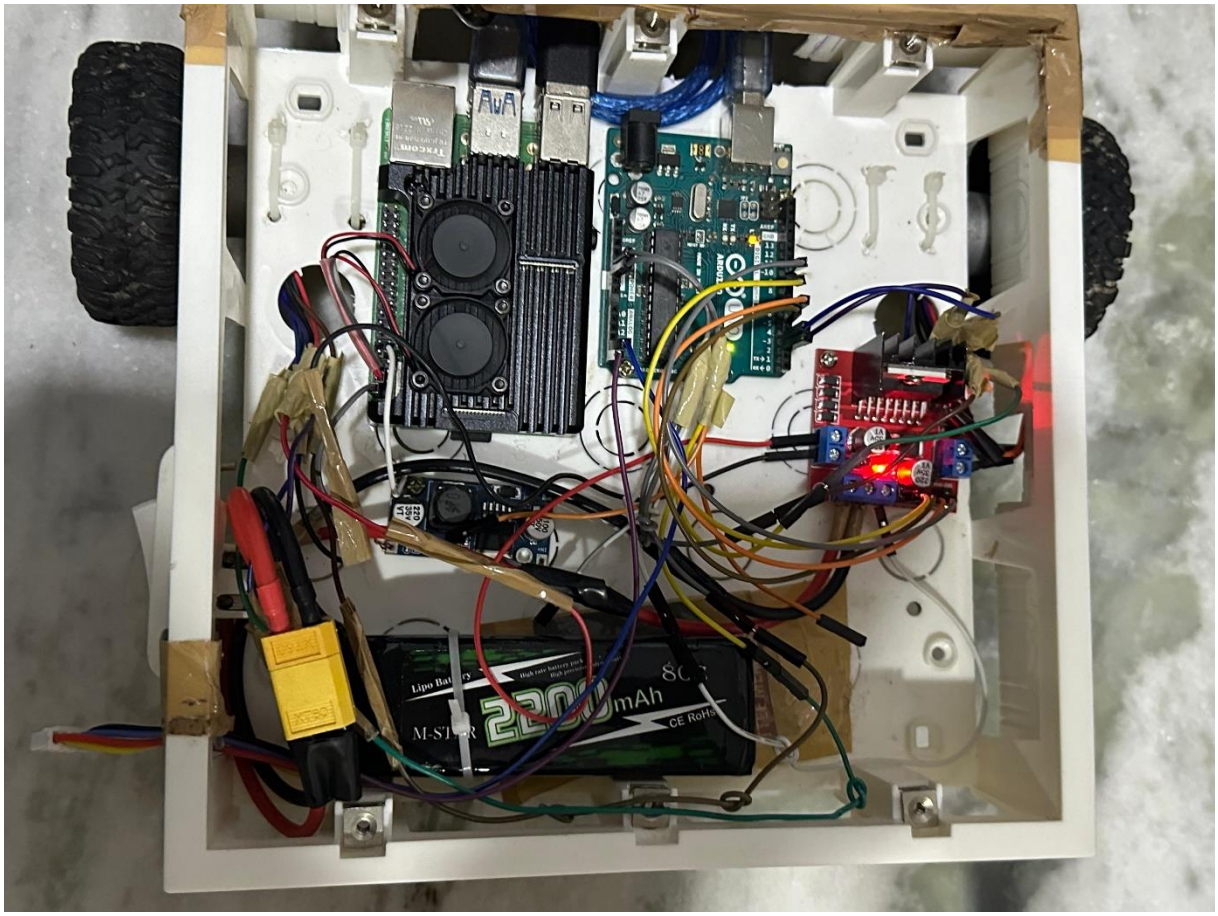


Fig.6: System design of ROS2 controlled robot.

5. CODING & IMPLEMENTATION

All the code mentioned below is available in GitHub.

https://github.com/srikanth-kandi/my_bot

GPIO Pins Configuration of Smart Localizer

Position of the robot 2 wheels back side and caster wheel facing me (l298n driver also facing me).

IN1 - D9

IN2 - D5

IN3 - D6

IN4 - D10

OUT1 - Left Motor Red

OUT2 - Left Motor Black

OUT3 - Right Motor Red

OUT4 - Right Motor Black

Arduino Pins

A4 - Left Motor Blue

A5 - Left Motor Violet

D2 - Right Motor Violet

D3 - Right Motor Blue

D5 - Motor Driver IN2

D6 - Motor Driver IN3

D9 - Motor Driver IN1

D10 - Motor Driver IN4

This pins configuration currently works both simulation and real robot.

Process to run slam_toolbox on Simulator

Clone https://github.com/srikanth-kandi/my_bot.git to ~/dev_ws/src on host system

Run the **colcon build --symlink-install**

Install slam_toolbox using **sudo apt install ros-foxy-slam-toolbox**

open 3 tabs in terminal

tab1 - ~/dev_ws$ **source install/setup.bash**

~/dev_ws$ **ros2 launch my_bot launch_sim.launch.py world:=./src/my_bot/worlds/obstacles.world**

tab2 - ~/dev_ws$ **rviz2 -d ./src/my_bot/config/slam_bot.rviz**

tab3 - ~/dev_ws$ **source install/setup.bash**

~/dev_ws$ **ros2 launch my_bot online_async_launch.py params_file:=./src/my_bot/config/mapper_params_online_async.yaml use_sim_time:=true**

Set the fixed frame to map in Rviz only after running tab3 slam command

For the Navigation

~/dev_ws$ **source install/setup.bash**

~/dev_ws$ **ros2 launch my_bot navigation_launch.py use_sim_time:=true**

and specifiy the goal pose estimate to move the robot to destination

After creating map in slam to save that map open new tab in terminal and enter

~$ **mkdir maps**

~$ **ros2 run nav2_map_server map_saver_cli -f maps/new_map_name**

Process to run SLAM on real robot

clone https://github.com/srikanth-kandi/my_bot.git to ~/robot_ws /src on Raspberry Pi

Run **colcon build --symlink-install** to install all dependencies

open 2 tabs in terminal window via SSH

tab1 - ~/robot_ws$ **source install/setup.bash**

~/robot_ws$ **ros2 launch my_bot launch_robot.launch.py**

tab2 - ~/robot_ws$ **source install/setup.bash**

~/robot_ws$ **ros2 launch my_bot ld19.launch.py**

open new terminal window and open 4 tabs on laptop

tab1 - ~/dev_ws$ **source install/setup.bash**

~/dev_ws$ **rviz2 -d ./src/my_bot/config/main.rviz**

turn on slam

tab2 - ~/dev_ws$ **source install/setup.bash**

~/dev_ws$ **ros2 launch my_bot online_async_launch.py params_file:=./src/my_bot/config/mapper_params_online_async.yaml use_sim_time:=false**

turn on joystick

tab3 - ~/dev_ws$ **source install/setup.bash**

~/dev_ws$ **ros2 launch my_bot joystick.launch.py**

set the fixed frame to map only after running tab2 slam command

For the Navigation

tab4 - ~/dev_ws$ **source install/setup.bash**

~/dev_ws$ **ros2 launch my_bot navigation_launch.py use_sim_time:=false**

and specify the goal pose estimate to move the robot to destination

joystick.launch.py

```
from launch import LaunchDescription
from launch_ros.actions import Node
from launch.substitutions import LaunchConfiguration
from launch.actions import DeclareLaunchArgument

import os
from ament_index_python.packages import get_package_share_directory

def generate_launch_description():
    use_sim_time = LaunchConfiguration('use_sim_time')

    joy_params = os.path.join(get_package_share_directory('my_bot'),'config','joystick.yaml')

    joy_node = Node(
            package='joy',
            executable='joy_node',
            parameters=[joy_params, {'use_sim_time': use_sim_time}],
        )

    teleop_node = Node(
            package='teleop_twist_joy',
            executable='teleop_node',
```

```
        name='teleop_node',
        parameters=[joy_params, {'use_sim_time': use_sim_time}],
        remappings=[('/cmd_vel','/cmd_vel_joy')]
    )

# twist_stamper = Node(
#         package='twist_stamper',
#         executable='twist_stamper',
#         parameters=[{'use_sim_time': use_sim_time}],
#         remappings=[('/cmd_vel_in','/diff_cont/cmd_vel_unstamped'),
#                     ('/cmd_vel_out','/diff_cont/cmd_vel')]
#     )


return LaunchDescription([
    DeclareLaunchArgument(
        'use_sim_time',
        default_value='false',
        description='Use sim time if true'),
    joy_node,
    teleop_node,
    # twist_stamper
])
```

launch_robot.launch.py

```
import os

from ament_index_python.packages import get_package_share_directory
```

```
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription, TimerAction
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import Command
from launch.actions import RegisterEventHandler
from launch.event_handlers import OnProcessStart

from launch_ros.actions import Node



def generate_launch_description():


    # Include the robot_state_publisher launch file, provided by our own package. Force sim time to be enabled
    # !!! MAKE SURE YOU SET THE PACKAGE NAME CORRECTLY !!!

    package_name='my_bot' #<--- CHANGE ME

    rsp = IncludeLaunchDescription(
                PythonLaunchDescriptionSource([os.path.join(
                    get_package_share_directory(package_name),'launch','rsp.launch.py'
                )]), launch_arguments={'use_sim_time': 'false', 'use_ros2_control': 'true'}.items()
    )

    # joystick = IncludeLaunchDescription(
    #             PythonLaunchDescriptionSource([os.path.join(
    #                 get_package_share_directory(package_name),'launch','joystick.launch.py'
```

```
    #            )])
    # )

    twist_mux_params =
os.path.join(get_package_share_directory(package_name),'config','twist_mux.yaml')
    twist_mux = Node(
            package="twist_mux",
            executable="twist_mux",
            parameters=[twist_mux_params],
            remappings=[('/cmd_vel_out','/diff_cont/cmd_vel_unstamped')]
        )

    robot_description = Command(['ros2 param get --hide-type /robot_state_publisher
robot_description'])

    controller_params_file =
os.path.join(get_package_share_directory(package_name),'config','my_controllers.yaml')

    controller_manager = Node(
        package="controller_manager",
        executable="ros2_control_node",
        parameters=[{'robot_description': robot_description},
                    controller_params_file]
    )

    delayed_controller_manager = TimerAction(period=3.0, actions=[controller_manager])

    diff_drive_spawner = Node(
        package="controller_manager",
        executable="spawner.py",
        arguments=["diff_cont"],
```

```
    )

    delayed_diff_drive_spawner = RegisterEventHandler(
        event_handler=OnProcessStart(
            target_action=controller_manager,
            on_start=[diff_drive_spawner],
        )
    )

    joint_broad_spawner = Node(
        package="controller_manager",
        executable="spawner.py",
        arguments=["joint_broad"],
    )

    delayed_joint_broad_spawner = RegisterEventHandler(
        event_handler=OnProcessStart(
            target_action=controller_manager,
            on_start=[joint_broad_spawner],
        )
    )

    # Code for delaying a node (I haven't tested how effective it is)
    #
    # First add the below lines to imports
    # from launch.actions import RegisterEventHandler
    # from launch.event_handlers import OnProcessExit
    #
    # Then add the following below the current diff_drive_spawner
    # delayed_diff_drive_spawner = RegisterEventHandler(
```

```
    #     event_handler=OnProcessExit(
    #         target_action=spawn_entity,
    #         on_exit=[diff_drive_spawner],
    #     )
    # )
    #
    # Replace the diff_drive_spawner in the final return with delayed_diff_drive_spawner

    # Launch them all!
    return LaunchDescription([
        rsp,
        # joystick,
        twist_mux,
        delayed_controller_manager,
        delayed_diff_drive_spawner,
        delayed_joint_broad_spawner
    ])
```

launch_sim.launch.py

```
import os

from ament_index_python.packages import get_package_share_directory



from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource

from launch_ros.actions import Node
```

```
def generate_launch_description():


    # Include the robot_state_publisher launch file, provided by our own package. Force sim time to be enabled
    # !!! MAKE SURE YOU SET THE PACKAGE NAME CORRECTLY !!!

    package_name='my_bot' #<--- CHANGE ME

    rsp = IncludeLaunchDescription(
                PythonLaunchDescriptionSource([os.path.join(
                    get_package_share_directory(package_name),'launch','rsp.launch.py'
                )]), launch_arguments={'use_sim_time': 'true', 'use_ros2_control': 'true'}.items()
    )

    joystick = IncludeLaunchDescription(
                PythonLaunchDescriptionSource([os.path.join(
                    get_package_share_directory(package_name),'launch','joystick.launch.py'
                )]), launch_arguments={'use_sim_time': 'true'}.items()
    )

    twist_mux_params = os.path.join(get_package_share_directory(package_name),'config','twist_mux.yaml')
    twist_mux = Node(
            package="twist_mux",
            executable="twist_mux",
            parameters=[twist_mux_params, {'use_sim_time': True}],
            remappings=[('/cmd_vel_out','/diff_cont/cmd_vel_unstamped')]
```

```
        )


    gazebo_params_file = os.path.join(get_package_share_directory(package_name),'config','gazebo_params.yaml')


    # Include the Gazebo launch file, provided by the gazebo_ros package
    gazebo = IncludeLaunchDescription(
                PythonLaunchDescriptionSource([os.path.join(
                    get_package_share_directory('gazebo_ros'), 'launch', 'gazebo.launch.py')]),
                    launch_arguments={'extra_gazebo_args': '--ros-args --params-file ' + gazebo_params_file}.items()
             )


    # Run the spawner node from the gazebo_ros package. The entity name doesn't really matter if you only have a single robot.
    spawn_entity = Node(package='gazebo_ros', executable='spawn_entity.py',
                        arguments=['-topic', 'robot_description',
                                   '-entity', 'my_bot'],
                        output='screen')


    diff_drive_spawner = Node(
        package="controller_manager",
        executable="spawner.py",
        arguments=["diff_cont"],
    )


    joint_broad_spawner = Node(
        package="controller_manager",
        executable="spawner.py",
        arguments=["joint_broad"],
    )
```

```
    # Code for delaying a node (I haven't tested how effective it is)
    #
    # First add the below lines to imports
    # from launch.actions import RegisterEventHandler
    # from launch.event_handlers import OnProcessExit
    #
    # Then add the following below the current diff_drive_spawner
    # delayed_diff_drive_spawner = RegisterEventHandler(
    #     event_handler=OnProcessExit(
    #         target_action=spawn_entity,
    #         on_exit=[diff_drive_spawner],
    #     )
    # )
    #
    # Replace the diff_drive_spawner in the final return with delayed_diff_drive_spawner

    # Launch them all!
    return LaunchDescription([
        rsp,
        joystick,
        twist_mux,
        gazebo,
        spawn_entity,
        diff_drive_spawner,
        joint_broad_spawner
    ])
```

ld19.launch.py

```
#!/usr/bin/env python3
from launch import LaunchDescription
from launch_ros.actions import Node

'''
Parameter Description:
---
- Set laser scan directon:
  1. Set counterclockwise, example: {'laser_scan_dir': True}
  2. Set clockwise,        example: {'laser_scan_dir': False}
- Angle crop setting, Mask data within the set angle range:
  1. Enable angle crop fuction:
    1.1. enable angle crop,  example: {'enable_angle_crop_func': True}
    1.2. disable angle crop, example: {'enable_angle_crop_func': False}
  2. Angle cropping interval setting:
  - The distance and intensity data within the set angle range will be set to 0.
  - angle >= 'angle_crop_min' and angle <= 'angle_crop_max' which is [angle_crop_min, angle_crop_max], unit is degress.
    example:
      {'angle_crop_min': 135.0}
      {'angle_crop_max': 225.0}
      which is [135.0, 225.0], angle unit is degress.
'''

def generate_launch_description():
  # LDROBOT LiDAR publisher node
  ldlidar_node = Node(
      package='ldlidar_stl_ros2',
      executable='ldlidar_stl_ros2_node',
      name='LD19',
```

```
    output='screen',
    parameters=[
      {'product_name': 'LDLiDAR_LD19'},
      {'topic_name': 'scan'},
      {'frame_id': 'base_laser'},
      {'port_name': '/dev/ttyUSB0'},
      {'port_baudrate': 230400},
      {'laser_scan_dir': True},
      {'enable_angle_crop_func': False},
      {'angle_crop_min': 135.0},
      {'angle_crop_max': 225.0}
    ]
)

# base_link to base_laser tf node
base_link_to_laser_tf_node = Node(
  package='tf2_ros',
  executable='static_transform_publisher',
  name='base_link_to_base_laser_ld19',
  arguments=['0','0','0.18','0','0','0','base_link','base_laser']
)



# Define LaunchDescription variable
ld = LaunchDescription()

ld.add_action(ldlidar_node)
ld.add_action(base_link_to_laser_tf_node)

return ld
```

rsp.launch.py

```
import os

from ament_index_python.packages import get_package_share_directory

from launch import LaunchDescription
from launch.substitutions import LaunchConfiguration, Command
from launch.actions import DeclareLaunchArgument
from launch_ros.actions import Node

import xacro


def generate_launch_description():

    # Check if we're told to use sim time
    use_sim_time = LaunchConfiguration('use_sim_time')
    use_ros2_control = LaunchConfiguration('use_ros2_control')

    # Process the URDF file
    pkg_path = os.path.join(get_package_share_directory('my_bot'))
    xacro_file = os.path.join(pkg_path,'description','robot.urdf.xacro')
    # robot_description_config = xacro.process_file(xacro_file).toxml()
    robot_description_config = Command(['xacro ', xacro_file, ' use_ros2_control:=',
use_ros2_control, ' sim_mode:=', use_sim_time])

    # Create a robot_state_publisher node
    params = {'robot_description': robot_description_config, 'use_sim_time': use_sim_time}
    node_robot_state_publisher = Node(
```

```
        package='robot_state_publisher',
        executable='robot_state_publisher',
        output='screen',
        parameters=[params]
    )


    # Launch!
    return LaunchDescription([
        DeclareLaunchArgument(
            'use_sim_time',
            default_value='false',
            description='Use sim time if true'),
        DeclareLaunchArgument(
            'use_ros2_control',
            default_value='true',
            description='Use ros2_control if true'),
        node_robot_state_publisher
    ])
```

robot.urdf.xacro

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro"  name="robot">

    <xacro:arg name="use_ros2_control" default="true"/>
    <xacro:arg name="sim_mode" default="false"/>

    <xacro:include filename="robot_core.xacro" />
```

```
    <xacro:if value="$(arg use_ros2_control)">
        <xacro:include filename="ros2_control.xacro" />
    </xacro:if>
    <xacro:unless value="$(arg use_ros2_control)">
        <xacro:include filename="gazebo_control.xacro" />
    </xacro:unless>

    <xacro:include filename="lidar.xacro" />

    <xacro:include filename="camera.xacro" />

    <!-- <xacro:include filename="depth_camera.xacro" /> -->

</robot>
```

robot_core.xacro

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro" >

    <xacro:include filename="inertial_macros.xacro"/>

    <material name="white">
        <color rgba="1 1 1 1" />
    </material>

    <material name="orange">
        <color rgba="1 0.3 0.1 1"/>
    </material>
```

```
<material name="blue">
   <color rgba="0.2 0.2 1 1"/>
</material>

<material name="black">
   <color rgba="0 0 0 1"/>
</material>

<material name="red">
   <color rgba="1 0 0 1"/>
</material>

<!-- BASE LINK -->

<link name="base_link">

</link>

<!-- BASE_FOOTPRINT LINK -->

<joint name="base_footprint_joint" type="fixed">
   <parent link="base_link"/>
   <child link="base_footprint"/>
   <origin xyz="0 0 0" rpy="0 0 0"/>
</joint>

<link name="base_footprint">
</link>
```

```
<!-- CHASSIS LINK -->

<joint name="chassis_joint" type="fixed">
   <parent link="base_link"/>
   <child link="chassis"/>
   <origin xyz="-0.1 0 0"/>
</joint>

<link name="chassis">
   <visual>
      <origin xyz="0.15 0 0.075"/>
      <geometry>
         <box size="0.3 0.3 0.15"/>
      </geometry>
      <material name="white"/>
   </visual>
   <collision>
      <origin xyz="0.15 0 0.075"/>
      <geometry>
         <box size="0.3 0.3 0.15"/>
      </geometry>
   </collision>
   <xacro:inertial_box mass="0.5" x="0.3" y="0.3" z="0.15">
      <origin xyz="0.15 0 0.075" rpy="0 0 0"/>
   </xacro:inertial_box>
</link>

<gazebo reference="chassis">
   <material>Gazebo/White</material>
</gazebo>
```

```
<!-- LEFT WHEEL LINK -->

<joint name="left_wheel_joint" type="continuous">
    <parent link="base_link"/>
    <child link="left_wheel"/>
    <origin xyz="0 0.175 0" rpy="-${pi/2} 0 0" />
    <axis xyz="0 0 1"/>
</joint>

<link name="left_wheel">
    <visual>
        <geometry>
            <cylinder radius="0.05" length="0.04"/>
        </geometry>
        <material name="blue"/>
    </visual>
    <collision>
        <geometry>
            <cylinder radius="0.05" length="0.04"/>
        </geometry>
    </collision>
    <xacro:inertial_cylinder mass="0.1" length="0.04" radius="0.05">
        <origin xyz="0 0 0" rpy="0 0 0"/>
    </xacro:inertial_cylinder>
</link>

<gazebo reference="left_wheel">
    <material>Gazebo/Blue</material>
</gazebo>
```

```
<!-- RIGHT WHEEL LINK -->

<joint name="right_wheel_joint" type="continuous">
    <parent link="base_link"/>
    <child link="right_wheel"/>
    <origin xyz="0 -0.175 0" rpy="${pi/2} 0 0" />
    <axis xyz="0 0 -1"/>
</joint>

<link name="right_wheel">
    <visual>
        <geometry>
            <cylinder radius="0.05" length="0.04"/>
        </geometry>
        <material name="blue"/>
    </visual>
    <collision>
        <geometry>
            <cylinder radius="0.05" length="0.04"/>
        </geometry>
    </collision>
    <xacro:inertial_cylinder mass="0.1" length="0.04" radius="0.05">
        <origin xyz="0 0 0" rpy="0 0 0"/>
    </xacro:inertial_cylinder>
</link>

<gazebo reference="right_wheel">
    <material>Gazebo/Blue</material>
</gazebo>
```

```
<!-- CASTER WHEEL LINK -->

<joint name="caster_wheel_joint" type="fixed">
    <parent link="chassis"/>
    <child link="caster_wheel"/>
    <origin xyz="0.24 0 0"/>
</joint>


<link name="caster_wheel">
    <visual>
        <geometry>
            <sphere radius="0.05"/>
        </geometry>
        <material name="black"/>
    </visual>
    <collision>
        <geometry>
            <sphere radius="0.05"/>
        </geometry>
    </collision>
    <xacro:inertial_sphere mass="0.1" radius="0.05">
        <origin xyz="0 0 0" rpy="0 0 0"/>
    </xacro:inertial_sphere>
</link>

<gazebo reference="caster_wheel">
    <material>Gazebo/Black</material>
    <mu1 value="0.001" />
```

```
    <mu2 value="0.001" />
  </gazebo>

</robot>
```

ros2_control.xacro

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">

  <xacro:unless value="$(arg sim_mode)">
    <ros2_control name="RealRobot" type="system">
      <hardware>
        <plugin>diffdrive_arduino/DiffDriveArduino</plugin>
        <param name="left_wheel_name">left_wheel_joint</param>
        <param name="right_wheel_name">right_wheel_joint</param>
        <param name="loop_rate">30</param>
        <param name="device">/dev/ttyACM0</param>
        <param name="baud_rate">57600</param>
        <param name="timeout">1000</param>
        <param name="enc_counts_per_rev">3436</param>
      </hardware>
      <joint name="left_wheel_joint">
        <command_interface name="velocity">
          <param name="min">-10</param>
          <param name="max">10</param>
        </command_interface>
        <state_interface name="velocity"/>
        <state_interface name="position"/>
      </joint>
```

```
        <joint name="right_wheel_joint">
            <command_interface name="velocity">
                <param name="min">-10</param>
                <param name="max">10</param>
            </command_interface>
            <state_interface name="velocity"/>
            <state_interface name="position"/>
        </joint>
    </ros2_control>
</xacro:unless>

<xacro:if value="$(arg sim_mode)">
    <ros2_control name="GazeboSystem" type="system">
        <hardware>
            <plugin>gazebo_ros2_control/GazeboSystem</plugin>
        </hardware>
        <joint name="left_wheel_joint">
            <command_interface name="velocity">
                <param name="min">-10</param>
                <param name="max">10</param>
            </command_interface>
            <state_interface name="velocity"/>
            <state_interface name="position"/>
        </joint>
        <joint name="right_wheel_joint">
            <command_interface name="velocity">
                <param name="min">-10</param>
                <param name="max">10</param>
            </command_interface>
            <state_interface name="velocity"/>
```

```
            <state_interface name="position"/>
          </joint>
      </ros2_control>
   </xacro:if>

   <gazebo>
      <plugin name="gazebo_ros2_control" filename="libgazebo_ros2_control.so">
         <parameters>$(find my_bot)/config/my_controllers.yaml</parameters>
         <parameters>$(find my_bot)/config/gaz_ros2_ctl_use_sim.yaml</parameters>
      </plugin>
   </gazebo>
</robot>
```

twist_mux.yaml

```
twist_mux:
  ros__parameters:
    topics:
      navigation:
        topic   : cmd_vel
        timeout : 0.5
        priority: 10
      joystick:
        topic   : cmd_vel_joy
        timeout : 0.5
        priority: 100
```

IMPLEMENTATION

Table1: Error Rate of the Trained Sentences

Sentences trained	**Error rate**
Refer to IBM lab	0%
Refer to IT workshop lab	0%
Refer to Project lab	0%
Refer to 106	0%
Refer to 102	0%
Refer to 105	0%
Refer to 107	0%

Table2: Trails to Get Accurate Accuracy

Commands	Trails	Correct time
Refer to IBM lab	5	5
Refer to IT workshop lab	5	5
Refer to Project lab	5	5
Refer to 106	5	5
Refer to 102	5	5
Refer to 105	5	5
Refer to 107	5	5

6. SYSTEM TESTING

6.1 Overview of Testing

Software testing involves comparing the programme to system requirements and user-provided requirements. Programme code is tested at the module level or at the phase level of the software development life cycle. The two parts of software testing are verification and validation.

6.2 Types of Tests

6.2.1 Unit Testing

Unit testing is a level of software testing in which individual programme modules or units are examined. Verifying that every software component operates as intended is the goal. The smallest testable component of any piece of software is called a unit. It often has one or more inputs and one or more outputs.

6.2.2 Integration Testing

Software is tested at the integration testing level, which combines testing of individual units with group testing. This level of testing aims to identify errors in the way integrated components interact with one another. Integration testing is aided by the usage of test stubs and drivers.

7. RESULTS

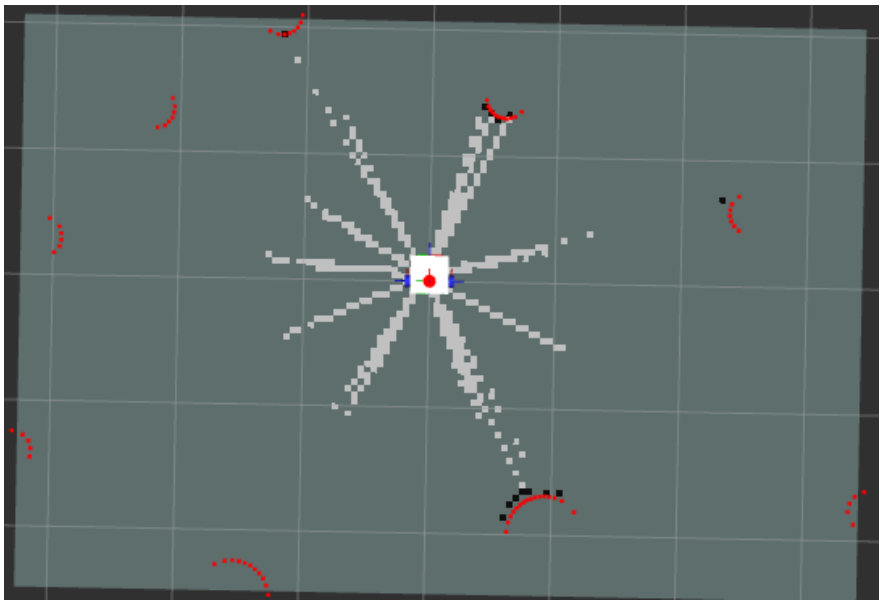


Fig.7: Before giving input to Smart Localizer in SLAM mode

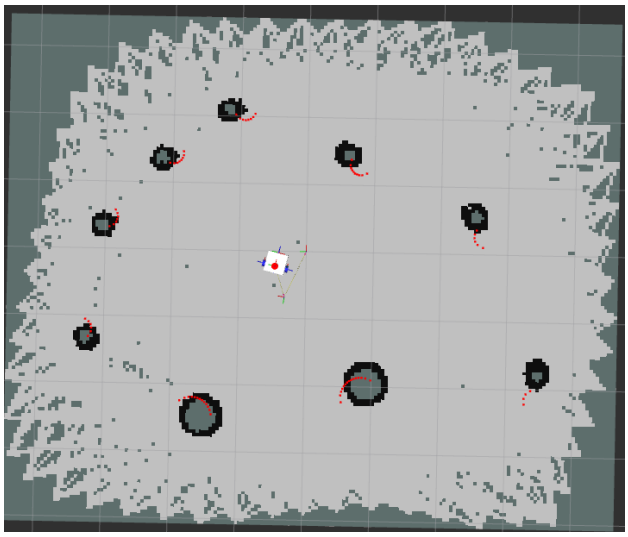
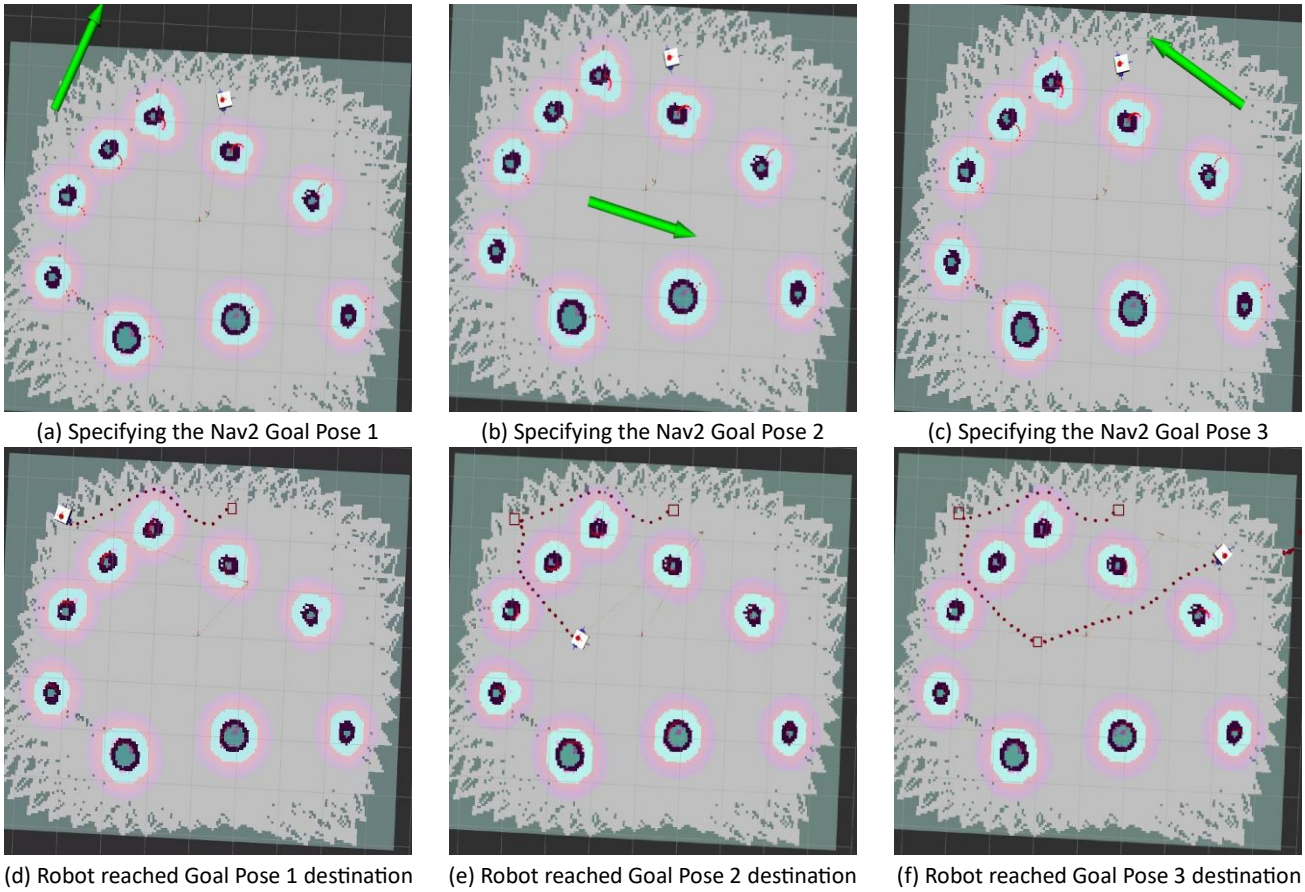


Fig. 8: After the complete map creation in SLAM mode



(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

Figure 9: Destination Path of Smart Localizer in specifying multiple Navigation 2 Goal Poses on Waypoint mode

8. CONCLUSION

The emergence of robots in modern times has made life easier and more demanding in numerous sectors. In this project, the field of robotics is investigated, and a robotic trolley equipped with many functionalities is proposed. It was proposed a mobile robot platform with a fixed two-wheel configuration chassis and having an electronic system designed around the Motor driver and Arduino Uno Interfaces. The designed model i.e smart Localizer detects any obstacles encountered in front of it. The usage of this equipment requires no prior training and reduces the amount of manual labor. They would become more efficient, accurate, and time saving by utilizing this software. Smart Localizer the low-cost solution, highly reliable and extendable to do small works.

9. 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.

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

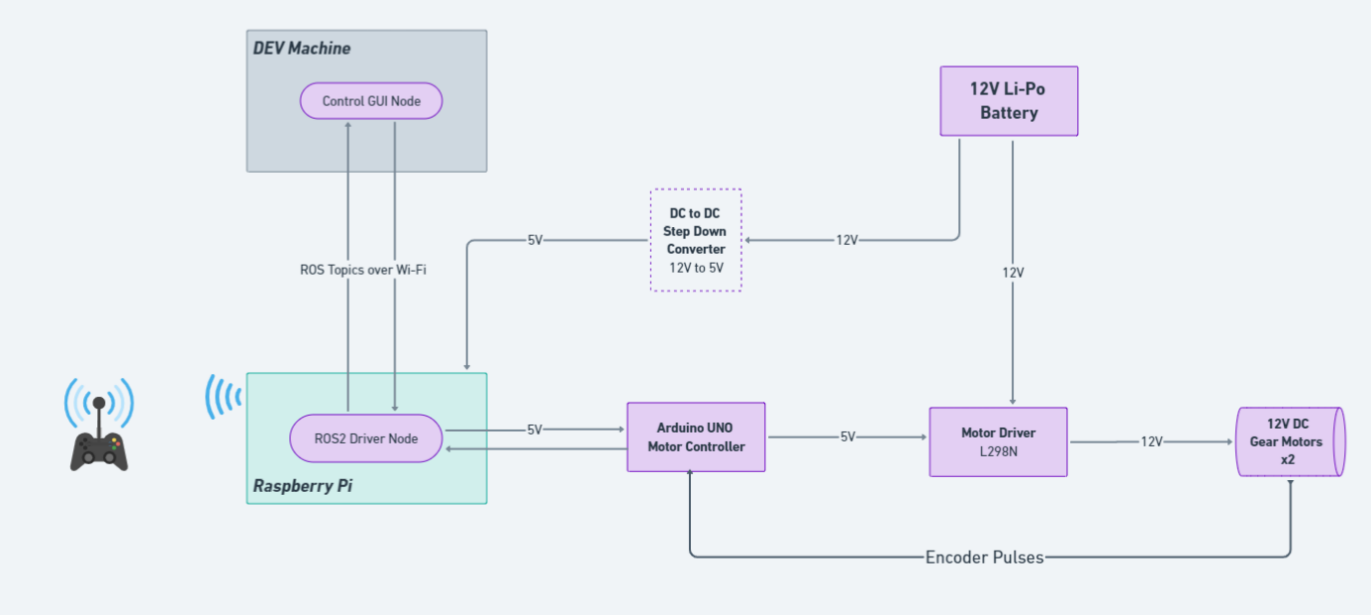


Fig. 1: Overall Structure of Smart Localizer

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

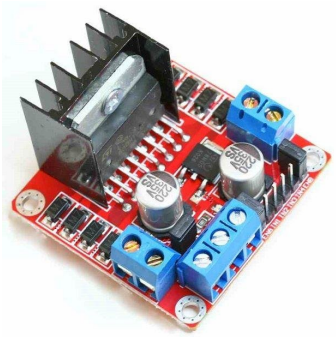


Fig. 2: Top View of L298N Motor Driver

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.

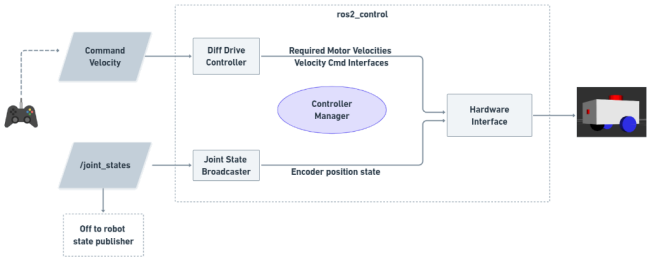


Fig. 4: Overview of ROS2 Control

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

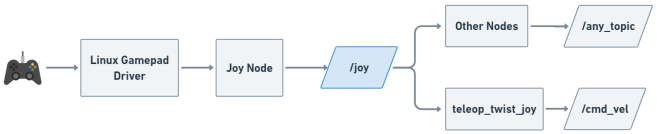


Fig. 5: Flow of Twist messages from Gamepad

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.

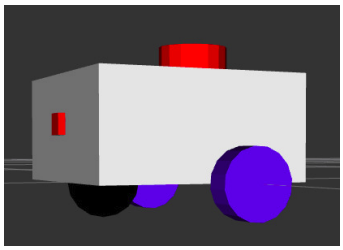


Fig. 6: 3D Model of Smart Localizer

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`.

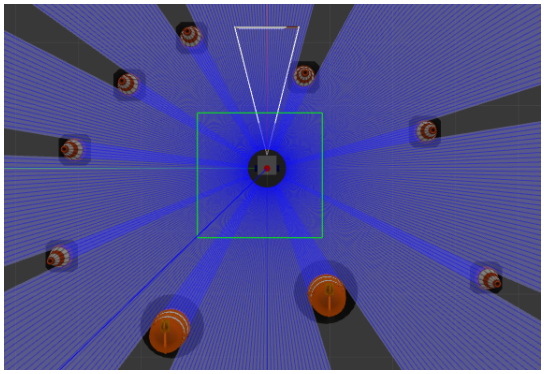


Fig. 7: Spawning robot in Gazebo world

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.

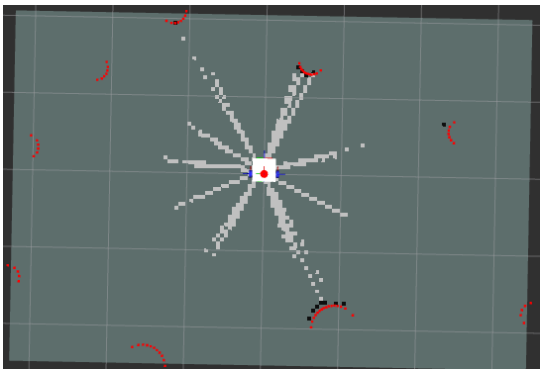


Fig. 8: Initial map created before traversing in Gazebo world

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.

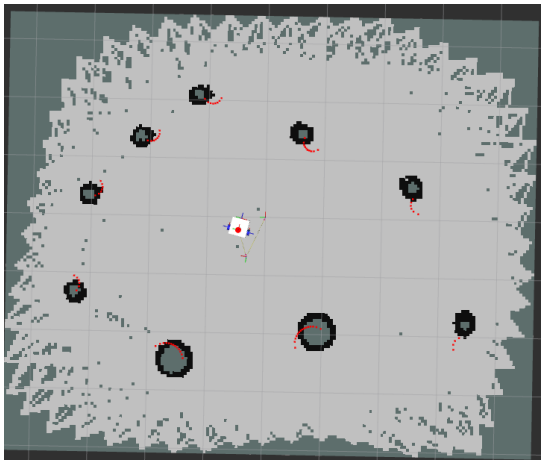


Fig. 9: Completed map after traversing in Gazebo world

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.

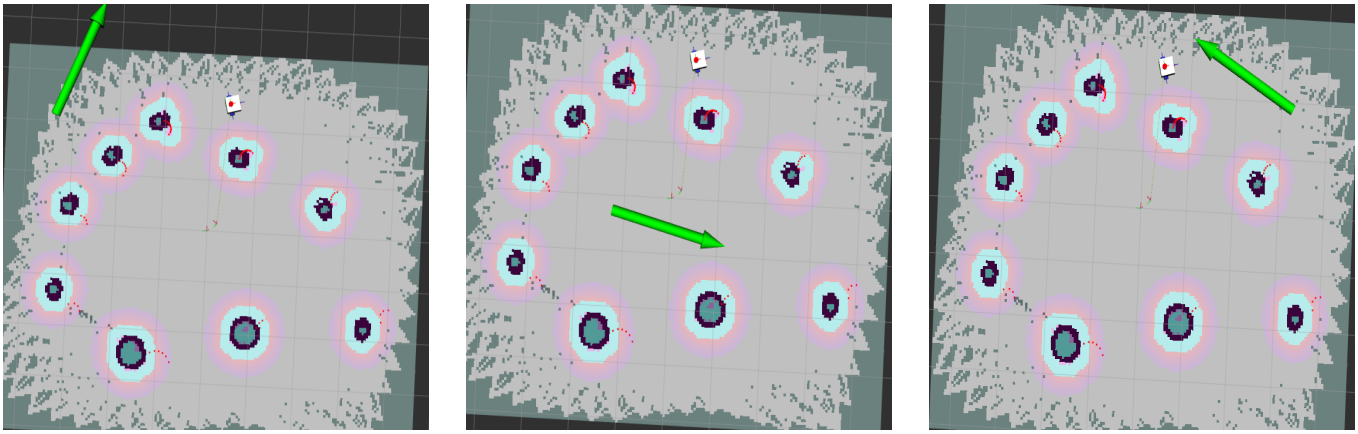
Nav2 stack provides a plugin for Rviz2, we can add a Navigation 2 panel with a Waypoint mode option after activating it, we need to specify multiple waypoints or destinations required for the robot to reach. As shown in Figures 10a, 10b and 10c we have specified 3 random waypoints for the Smart Localizer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

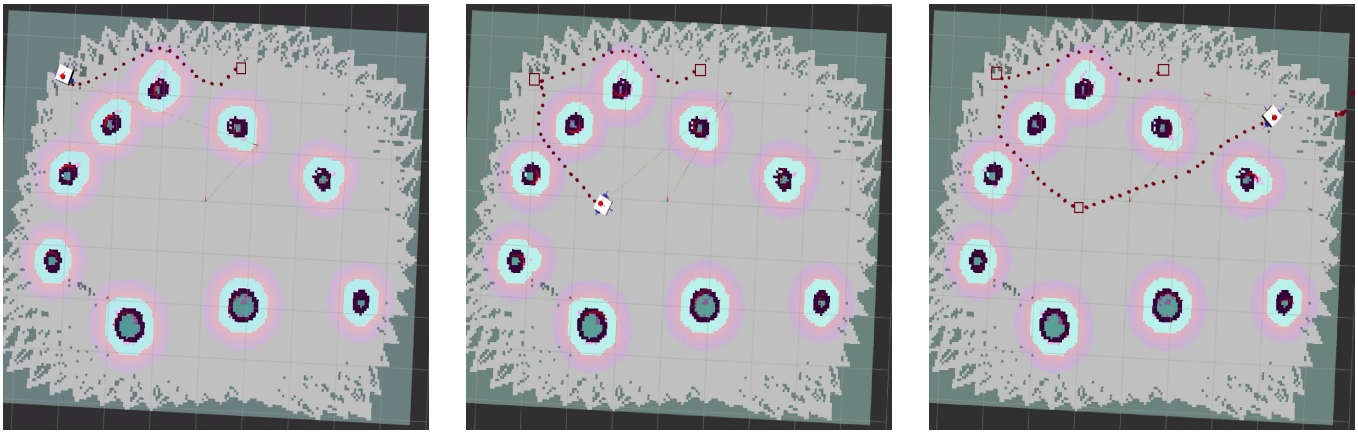
[8]https://github.com/ldrobotSensorTeam/ldlidar_stl_ros2

[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

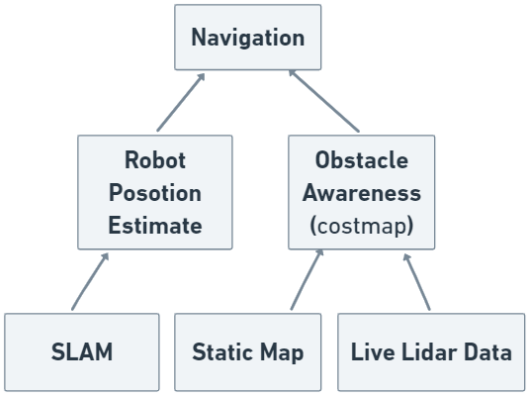


Fig. 11: Flow of Navigation 2

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.

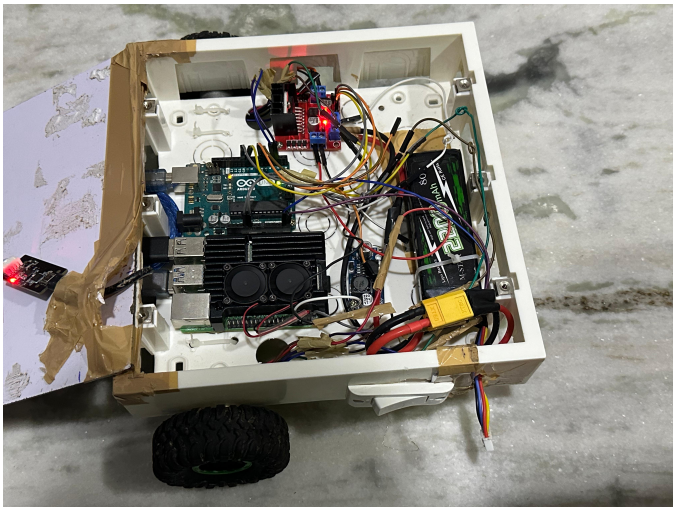


Fig. 12: Top view of Smart Localizer while opened

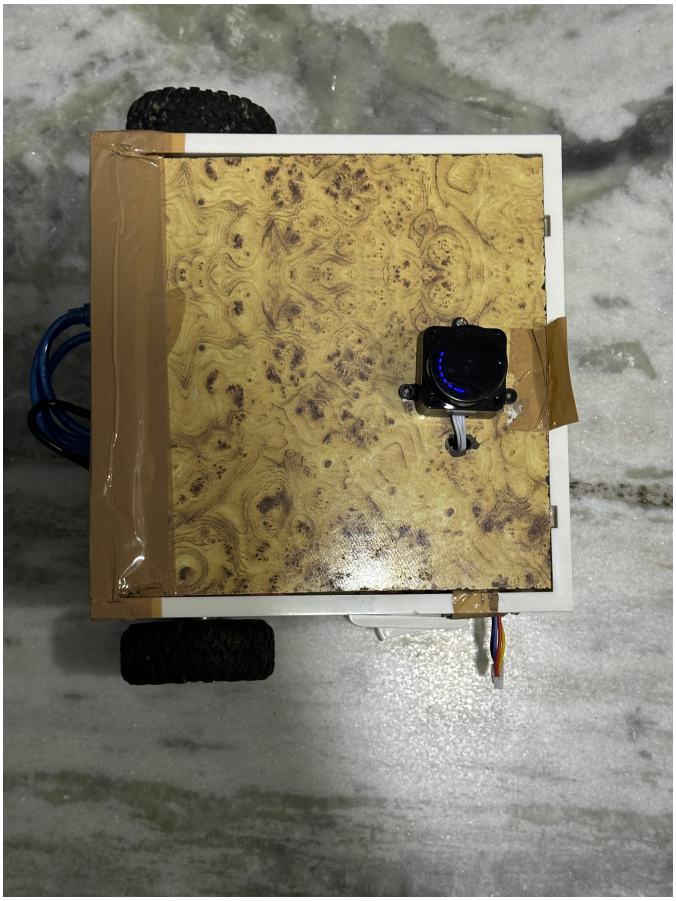


Fig. 13: View of Smart Localizer while closed

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.