IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 11, November 2024

DOI: 10.17148/IJARCCE.2024.131106

Design and Implementation of Visual SLAM Delivery Robot

Amal Shaji[1], Aslam Nassar[1], Rohan Roy[1], Syamjin M[1],

Dr. Kiran R[1]

Govt. Engineering College Idukki, Kerala, India[1]

Abstract: This paper presents the design, development and implementation of a new autonomous delivery robot employing Simultaneous Localization and Mapping (SLAM) technology. By integrating depth cameras, 2D LiDAR, and Inertial Measurement Unit (IMU) sensors, the robot constructs a real-time map of its environment and identifies obstacles. Utilizing the ROS2 navigation stack, the robot plans optimal routes and avoids obstacles, thereby facilitating efficient and autonomous delivery. The implemented hardware was able to successfully navigate through a test terrain with instantaneous localization and map generation, when target points for the delivery robot are provided via desktop software.

Keywords: Visual SLAM, robot navigation, guidance, sensor fusion.

I. INTRODUCTION

The demand for efficient and automated delivery solutions has become increasingly crucial in the rapidly evolving technological landscape. Traditional delivery methods, often reliant on human labor, face challenges in scalability, cost, and efficiency. Autonomous delivery robots, equipped with advanced technologies such as Visual Simultaneous Localization and Mapping (Visual SLAM), present a promising alternative. VSLAM systems have matured over the past few years, and several frameworks have played a vital role in this development process [1].

With the baseline of VSLAM, researchers focused on improving the performance and precision of these systems. In this regard, Forster et al. [2] proposed a hybrid VO approach known as Semi-direct Visual Odometry (SVO) as a part of VSLAM architectures. Their method could merge feature-based and direct approaches to perform sensors' motion estimation and mapping tasks. SVO could work with both monocular and stereo cameras and was equipped with a pose refinement module to minimize re-projection errors. However, the main drawbacks of SVO are employing a short-term data association and the inability to perform loop closure detection and global optimization. LSD-SLAM [3] is another influential VSLAM method introduced by Engel et al., and contains tracking, depth map estimation, and map optimization threads. The method could reconstruct large-scale maps using its pose-graph estimation module and was equipped with global optimization and loop closure detection feat. The weakness of LSD-SLAM is its challenging initialization stage that requires all points in a plane, making it a computationally intensive approach. Mur-Artal et al. introduced two accurate indirect VSLAM approaches that have attracted the attention of many researchers so far: ORB-SLAM [4] and ORB-SLAM 2.0 [5]. These methods can accomplish localization and mapping in well-textured sequences and perform high performance position recognition using Oriented FAST and Rotated BRIEF (ORB) features. The first version of ORB-SLAM is able to compute both the camera position and the environment's structure using the key frames collected from camera locations. Although ORB-SLAM 2.0 can work with both monocular and stereo camera setups, it cannot be used for autonomous navigation due to reconstructing maps with unknown scales. Another drawback of this approach is its inability to work in textureless areas or environments with repetitive patterns. The most recent version of this framework, named ORB-SLAM 3.0, was proposed [6]. It works with various camera types, such as monocular, RGB-D and stereo-vision, and provides improved pose estimation outputs.

II. OBJECTIVES

This research focuses on the development of an autonomous delivery robot, aiming to provide a reliable and cost-effective solution for last-mile delivery. The robust framework should allow the robot to analyze its surroundings, selecting the most efficient route, and dynamically re-planning as needed. Primary objective of this work is to develop the software for mapping and navigation of the robot using the RoS2 framework that involves the following.

1. Software Development: Utilizing the ROS2 framework, that provides tools and libraries for robot control, navigation, and sensor processing.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 11, November 2024

DOI: 10.17148/IJARCCE.2024.131106

2. Mapping: Implementing algorithms like SLAM (Simultaneous Localization and Mapping) to enable the robot to build a real-time map of its environment.
3. Navigation: Designing algorithms that use the map and sensor data to plan and execute efficient paths to reach designated destinations, while avoiding obstacles.

III. METHODOLOGY AND DESIGN

The methodology follows an iterative and incremental approach, integrating hardware and software components in parallel while testing and validating at each stage. It emphasizes modularity, scalability, and flexibility to accommodate changes and optimizations as the work progresses. Collaboration and communication among team members, adherence to best practices, and a systematic approach to problem-solving are fundamental elements ensuring the success of the implementation. This involves the following subsections.

System Architecture Design: Designing the system architecture involves defining the overall structure, components, and interactions. It includes deciding on hardware components, sensor integration, software modules, and their interconnections within the ROS2 framework as described below.

1. Hardware Integration: This phase deals with integrating hardware components such as sensors (2D LiDAR, depth cameras, IMU), actuators, microcontrollers, and other peripherals onto the robot's chassis. It involves physical assembly, wiring, and ensuring seamless interaction between hardware components.
2. Software Development: Developing software for mapping, navigation, and user control involves several stages. Implementing SLAM algorithms for mapping and localization using ROS2, designing control algorithms for navigation, and creating user-friendly interfaces for remote control and monitoring are key components.
3. Testing and Validation: Rigorous testing is essential to validate system functionalities. This includes testing sensor accuracy, mapping accuracy, navigation precision, and user interface usability. Validation is crucial to ensure that the system meets defined requirements and performs reliably in varied scenarios.
4. Iterative Improvement: Continuous refinement based on feedback and testing results is vital. Iterative cycles involve revisiting design aspects, making enhancements, and optimizing algorithms for better performance and reliability.

3. 1 Block Level Design of the Hardware

For an autonomous delivery robot implementation utilizing Visual SLAM with ROS2, several key hardware components are essential to enable perception, navigation, and control capabilities, as shown in Figure 1. Visual SLAM is central to this work, leveraging depth cameras, 2D LiDAR, and Inertial Measurement Unit (IMU) sensors to construct a real-time map of the environment while simultaneously determining the robot's position within it. This dynamic mapping capability enables the robot to navigate complex environments with precision, identifying and avoiding obstacles in its path. To ensure optimal route planning and efficient delivery, the work integrates the ROS2 navigation stack.

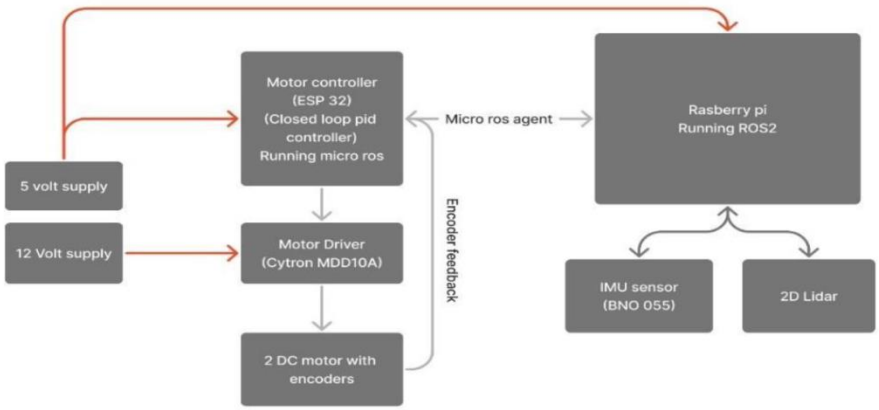

Figure 1: Block level diagram of autonomous delivery robot

DOI: 10.17148/IJARCCE.2024.131106

3.2 Software Design

ROS2 framework: The ROS2 software framework forms the backbone of the autonomous delivery robot, orchestrating its operation through a distributed and real-time approach. It facilitates communication between various components like sensors, actuators, and control algorithms, enabling a modular and flexible system. Each component operates as a separate ROS2 node, simplifying development and allowing for easy addition or removal of features. The DDS middleware within ROS2 ensures reliable and efficient data exchange between nodes, critical for real-time responsiveness. Visualization tools like RViz offer valuable insights into sensor data and robot state, aiding in performance monitoring and debugging. Package management further simplifies development by facilitating software installation and dependency management. Overall, ROS2 plays a crucial role in enabling the autonomous delivery robot to function effectively and efficiently, thanks to its distributed architecture, node-based design, communication middleware, visualization tools, and package management capabilities.

Micro-ROS: Micro-ROS is a streamlined version of ROS designed specifically for microcontrollers. Traditionally, ROS has been the framework of choice for complex robots equipped with powerful computing resources. By adapting ROS for microcontrollers like the ESP32, it paves the way for a new era of intelligent and interconnected robots. For small implementations with constrained resources, Micro-ROS condenses the essential functions of ROS, making it suitable for microcontrollers. Micro-ROS facilitates communication between microcontrollers and standard ROS systems running on devices like Raspberry Pi or PCs, enabling smaller robots to integrate into larger ROS ecosystems. Similar to full ROS, Micro-ROS allows microcontrollers to publish sensor data (such as temperature or distance) and subscribe to topics from other nodes, promoting collaboration and information sharing. Micro-ROS provides a more accessible development environment than traditional ROS, making it easier to program resource-limited microcontrollers for robotics applications. Micro-ROS bridges the gap between small robots and powerful ROS systems, fostering communication and collaboration within a broader robotic ecosystem.

Nav2 Navigation Stack: The Nav2 (Navigation2) navigation stack is a powerful and modular open source navigation framework primarily designed for mobile robots operating in ROS2 (Robot Operating System 2). It serves as a comprehensive and flexible solution for autonomous navigation, providing a collection of ROS2 packages and tools aimed at simplifying the development of navigation systems for robots. Nav2 is designed with a modular architecture, allowing developers to choose and swap out various components such as planners, controllers, and sensors. This modularity fosters flexibility and adaptability, enabling the stack to cater to diverse robot configurations and navigation. It supports different map representations like occupancy grids, cost maps, and voxel grids. This capability allows robots to perceive and navigate through their environments accurately by utilizing sensor data to create and update maps in real time. Nav2 offers a range of path planning algorithms and controllers, allowing robots to plan and execute trajectories efficiently. This includes global planners for long-term path planning and local planners for real time obstacle avoidance. Behavior trees are employed for defining complex navigation behaviors. Nav2 utilizes the BehaviorTree.CPP library to implement state machines and hierarchical behaviors, facilitating the creation of sophisticated navigation behaviors.

ROS2 Integration: As a ROS2-centric navigation stack, Nav2 seamlessly integrates with other ROS2 packages and tools. This integration enables communication between various components within the robot system and facilitates interoperability with different hardware and software modules.

Scalability and Optimization: The stack is designed to handle scalability and optimization challenges in real-world environments. It provides mechanisms for handling large-scale maps, optimizing computation, and addressing computational resource constraints.

IV. IMPLEMENTATION

4.1 Motor Controlling

A two-wheeled differential drive robot scoots around on two independently powered wheels. These robots zip and spin by controlling the speed and direction of each wheel. To move forward, both wheels whiz in the same direction. For turns, one wheel slows down or reverses, making the robot pivot around a central point. This simple design makes them perfect for navigating tight spaces, like a robot vacuum cleaner cleaning under your couch. However, their two-wheel drive can be a handful on bumpy terrain, and they may struggle to climb over obstacles. The motor control is the brain of the operation. It takes user input, translates it into motor commands, and implements control algorithms to achieve desired performance. Here's a breakdown of the key software aspects:

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 11, November 2024

DOI: 10.17148/IJARCCE.2024.131106

User Input and Velocity Mapping: The software receives user input, which could be joystick movement, button presses, or commands from a serial interface. Velocity Mapping translates this user input into a desired motor speed or direction. This can be a simple linear mapping (e.g., joystick up indicating higher speed) or a more complex function for finer control. For instance, an exponential mapping could provide more precise control at low speeds and faster response at high joystick positions.

Motor Control: Implementation of the motor control mechanism can be as shown in Figure 2. The motor direction (forward/reverse) can be set independently. Motor speed can be controlled using techniques like Pulse Width Modulation (PWM). Encoder data can be read using closed-loop control.

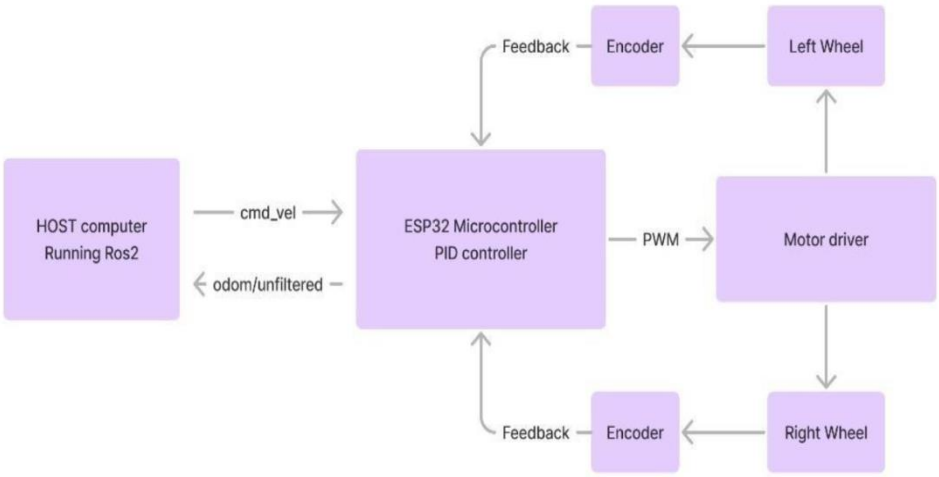

Figure 2: Implementation of the motor control mechanism.

4.2 Closed Loop Configuration

An open-loop system would be like setting the throttle and hoping for the best. In reality, even tiny differences in motor power or uneven floors can send the car veering off course. Closed-loop control acts like a skilled driver. Sensors on the wheels, like tiny odometers, keep track of actual speeds. This feedback is fed back to the robot's "brain," which compares it to the desired speeds. If there's a mismatch, the controller adjusts motor commands in real-time. This ensures both wheels move exactly as planned, keeping the robot on a straight line. But a closed-loop isn't just about going straight while making sharp turns. In open-loop, even a slight power difference between the wheels could result in a wobbly turn. Closed-loop measures each wheel's rotation and fine-tunes motor commands to achieve the perfect speed difference for a crisp turn. This constant feedback loop extends beyond basic movement. When the robot encounters a bump. Sensors can detect this change, and the controller can adjust motor commands to compensate. This adaptability allows the robot to navigate uneven terrain more effectively. In short, closed-loop control transforms a basic robot into a precise and responsive machine, allowing it to move with confidence in its surroundings. For precise control, especially in applications requiring constant speed or position, the Proportional Integral Derivative (PID) controller is used as shown in Figure 3.

The PID controller continuously monitors the difference between the desired speed (set point) and the actual speed measured by the encoder (feedback). Based on this difference, the controller adjusts the motor's output (via PWM) to minimize the error and achieve the desired speed. The Proportional term reacts to the current error, making immediate adjustments. The Integral term considers accumulated error over time, correcting for long-term deviations. The Derivative term anticipates future errors based on the rate of change of the error, providing smoother control.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 11, November 2024

DOI: 10.17148/IJARCCE.2024.131106

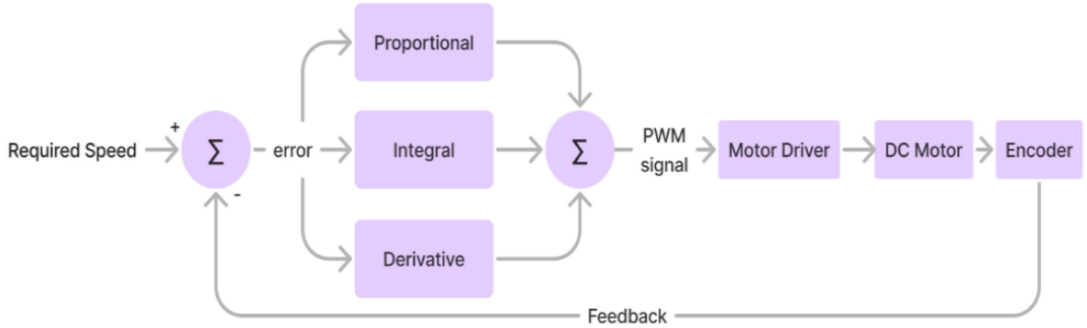

Figure 3: Top level design of PID controller

4.3 Robot Localization

Robot Positioning with Wheel Odometry, IMU, and Extended Kalman Filter (EKF): Robots need to know their location (x, y, and orientation) to navigate effectively. This can be tricky, especially for mobile robots that move around. This can be achieved using sensor fusion with EKF. The data from two sensors can be combined in the following way.

Wheel Odometry estimates robot movement based on wheel rotations. It is good for tracking short-term motion but accumulates errors over time (wheel slip, uneven terrain) as explained in Figure 4, where

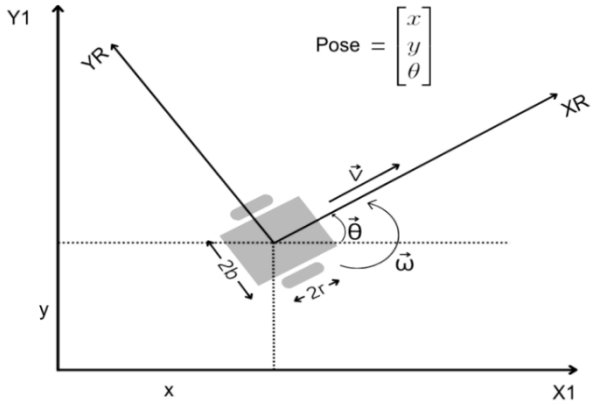

Figure 4: Layout of the wheel odometry

x: represents the forward and backward position of the robot along a horizontal axis.
y: represents the sideways position of the robot along a horizontal axis.
θ: represents the robot's orientation.
$2b$: is the distance between the two wheels.
r: is the radius of the robot's wheels.
v: is the linear velocity of the robot.
ω: is the angular velocity of the robot.

Thus the wheel odometry equation can be represented as

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} x(t-\Delta t) \\ y(t-\Delta t) \\ \theta(t-\Delta t) \end{bmatrix} + \Delta t \cdot \begin{bmatrix} \frac{r}{2} \cdot \cos\theta(t-\Delta t) & \frac{r}{2} \cdot \cos\theta(t-\Delta t) \\ \frac{r}{2} \cdot \sin\theta(t-\Delta t) & \frac{r}{2} \cdot \sin\theta(t-\Delta t) \\ \frac{r}{2 \cdot b} & -\frac{r}{2 \cdot b} \end{bmatrix} \cdot \begin{bmatrix} \dot{\psi}_R(t-\Delta t) \\ \dot{\psi}_L(t-\Delta t) \end{bmatrix}$$

IMU (Inertial Measurement Unit) provides information about the robot's acceleration and orientation (gyroscope and accelerometer). It's great for short- term orientation but suffers from drift (sensor noise accumulates over time). IMU data and wheel odometry is fused using the robot localization package in ROS2.

DOI: 10.17148/IJARCCE.2024.131106

Extended Kalman Filter (EKF) is a powerful sensor fusion technique. It takes noisy data from both sensors, predicts robot movement based on a motion model, and then corrects these predictions using sensor measurements. This combined approach offers a more accurate and reliable estimate of the robot's position and orientation.

Following three steps may be stated for the working of the sensor fusion system.

1. Prediction: The EKF predicts the robot's new state (position and orientation) based on previous odometry data and a motion model that considers factors like wheel speeds and elapsed time.
2. Measurement Update: Real-time data from the IMU (accelerometer and gyroscope) is incorporated. The EKF adjusts the predicted state based on how well it aligns with the IMU measurements.
3. Repeat: This prediction-update cycle happens continuously, providing a constantly refined estimate of the robot's location.

Overall, this sensor fusion approach using an EKF allows robots to overcome the limitations of individual sensors and achieve a more accurate and robust understanding of their location, paving the way for effective navigation and task execution.

4.4 Mapping using SLAM

SLAM equips you with two crucial abilities, 1. creating a map of the house as you explore it, and 2. keeping track of your exact location within that map. SLAM tackles two problems at once: building a map of an unknown environment (mapping) and figuring out where the robot is within that map (localization). Figure 5 shows the breakdown of SLAM robot mapping.

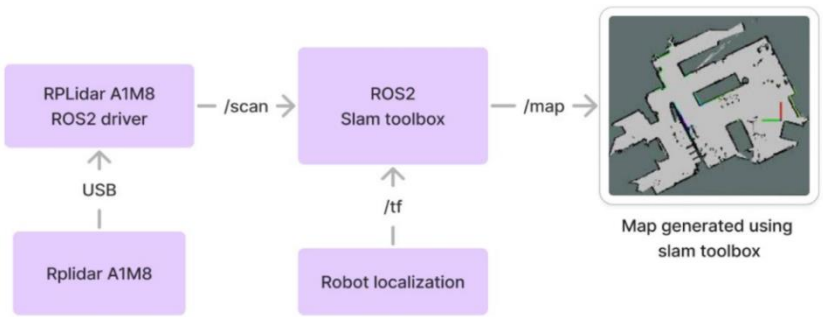

Figure 5: Block level design of map generation using SLAM

This can be achieved in the following steps

1. Sensors See the World: Robots rely on sensors like LiDAR (which use lasers to measure distances) or cameras to gather data about their surroundings. This data paints a picture of the environment.
2. SLAM Toolbox: The ROS 2 SLAM Toolbox empowers robots to build maps and localize themselves using sensors like LiDAR. This versatile package offers different SLAM algorithms for real-time performance or high accuracy, seamlessly integrating with ROS 2 for easy setup and communication with other robot components. By subscribing to sensor data and processing it through chosen algorithms, SLAM Toolbox constructs a map while simultaneously estimating the robot's location within that map, paving the way for effective robot navigation within ROS 2.

4.5 Autonomous Navigation

Building on a map created with SLAM or another localization system, ROS 2's Nav2 package orchestrates robot navigation. It dissects the journey into a global plan (considering the entire map) and a local plan (focusing on immediate surroundings with sensor data). Behavior trees then guide the robot through these waypoints, with a controller translating

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 11, November 2024

DOI: 10.17148/IJARCCE.2024.131106

them into motor commands. This modular and sensor-friendly framework within ROS 2 empowers robots to navigate complex environments and tackle various tasks efficiently. This is visually described in Figure 6.

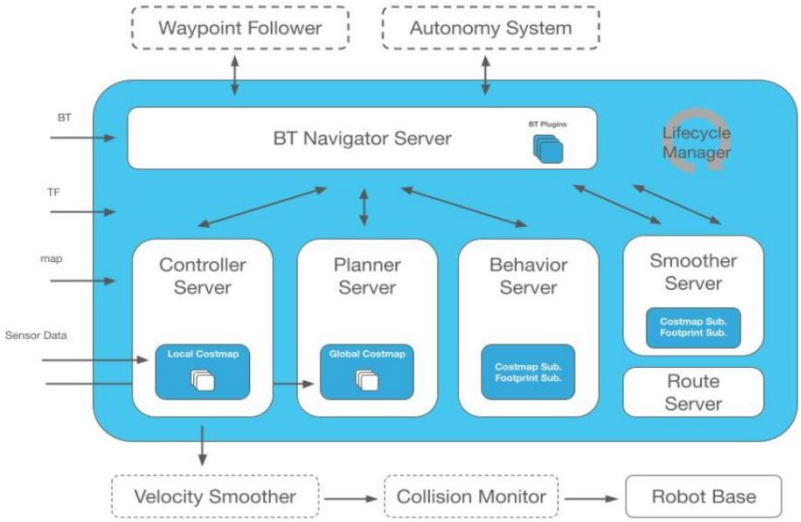

Figure 6: Block level design of autonomous navigation

V. RESULTS

In this work, a Raspberry Pi 3 model B based visual SLAM system for autonomous delivery applications was implemented. A finished final form of the implementation is shown in Figure 7. A closed loop PID motor controller for the accurate speed control of the motor is implemented. Localization of the robot by sensor fusion of wheel odometry and IMU sensor is achieved. The map of the environment of the robot is simultaneously generated using LiDAR and localization data. The robot successfully navigated autonomously from point A to point B on a test terrain. Figure 8 shows an instantaneous map generated by the SLAM. Figure 9 shows the trajectory traced by the robot during a test manoeuvre of autonomous navigation from point A to point B.

Figure 7: Assembled system

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 11, November 2024

DOI: 10.17148/IJARCCE.2024.131106

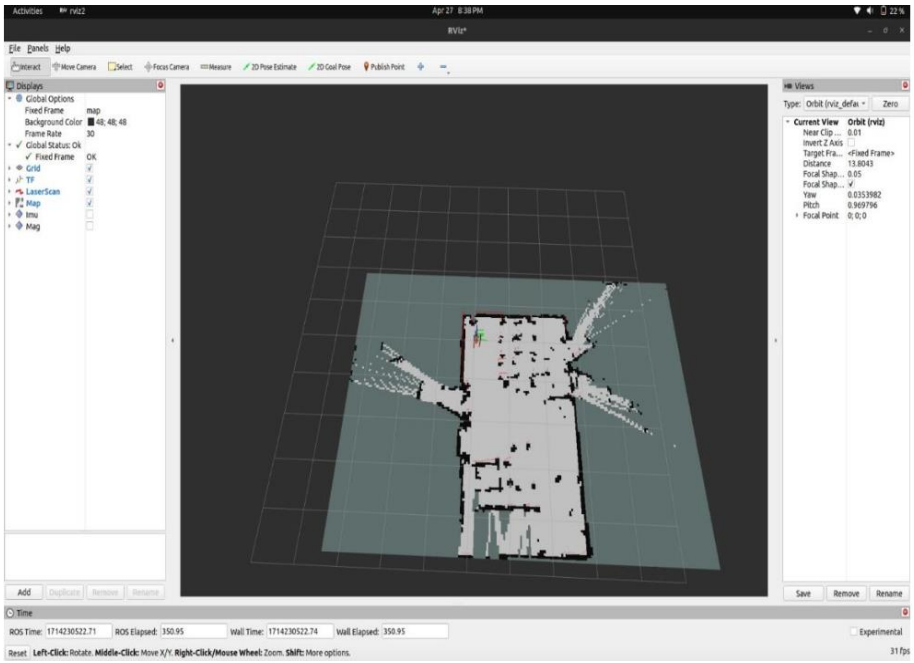

Figure 8: Map generated by the SLAM

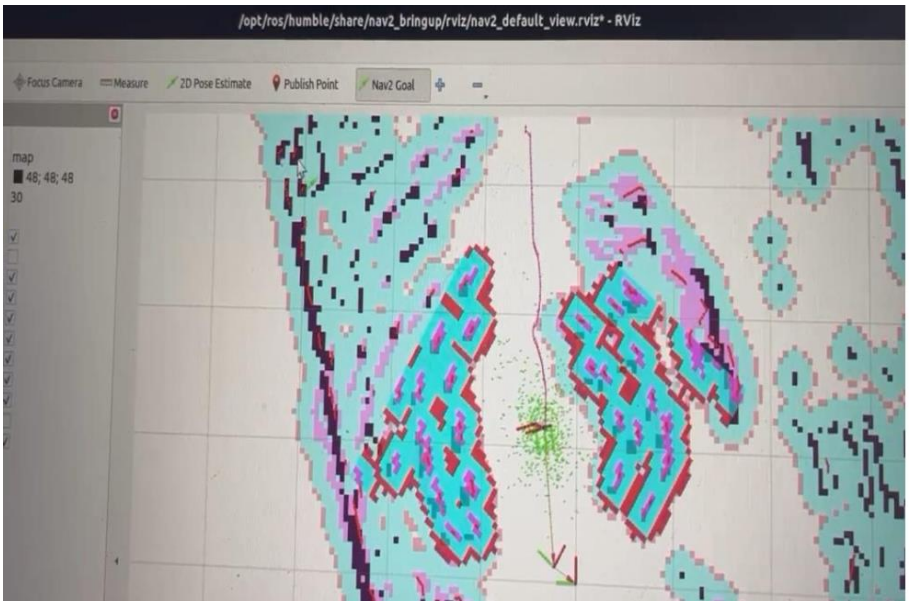

Figure 9: Autonomous navigation form point A to point B

VI. CONCLUSION

Visual SLAM is a technique that allows a robot to build a map of its surroundings and track its own position in that map. This work achieves a visual SLAM system using a camera to capture images of the environment and a computer to process those images. Sensor fusion is successfully achieved using Wheel Odometry, IMU, and Extended Kalman Filter data for localization. The computer uses the images to estimate the robot's position and orientation to build a map of the environment. Visual SLAM is important for autonomous delivery robots because it allows them to navigate complex environments. It can help robots navigate through cluttered and unfamiliar environments. By building a map of the environment, robots can plan routes that avoid obstacles and allow them to reach their destinations efficiently. It can help robots detect obstacles in their path and avoid them. This is important for safety and to ensure that the robot can deliver

DOI: 10.17148/IJARCCE.2024.131106

packages without damage. It can help robots find the correct recipient and deliver packages securely. This is important for customer satisfaction and to reduce the risk of theft or damage.

REFERENCES

[1]. A. Tourani, H. Bavle, J. Sanchez-Lopez, H. Voos, "Visual SLAM: What are the Current Trends and What to Expect?," *Journal of Sensors*, vol. 22, no. 23, 92-97, 2022.

[2]. C. Forster, M. Pizzoli, D. Scaramuzza, "SVO: Fast Semi-direct Monocular Visual Odometry," In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2014.

[3]. J. Engel, T. Schops, D. Cremers, "LSD-SLAM: Large-scale Direct Monocular SLAM," European Conference on Computer Vision, Springer, vol. 8690, pp. 834-849, 2014.

[4]. R. Mur-Artal, J. M. M. Montiel, J.D.Tardos, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147-1163, 2015.

[5]. R. Mur-Artal, J. D. Tardos, "ORB-SLAM2: An Open-source SLAM System for Monocular, Stereo, and RGB-D Cameras, *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255-1262, 2017.

[6]. C. Campos, R. Elvira, J. J. G. Rodriguez, J. M. M. Montiel, J. D.Tardos, ORB-SLAM3: An Accurate Open-source Library for Visual, Visual Inertial, and Multimap SLAM," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874–1890, 2021.

