GERMAN - MONGOLIAN INSTITUTE FOR RESOURCES AND TECHNOLOGY

The present work was submitted to

the German-Mongolian institute for Resources and Technology

Development of a Control System for a Street-Cleaning Robot

Bachelor Thesis

by

Munkhsuld Shijee

Study program: **Mechatronic engineer**

Student ID: **B2200388**

Supervisor 1 / Examiner **Ph.D., N.Odbileg**

Supervisor 2 / Examiner **Prof. Ph.D., P.Ariunbolor**

Ulaanbaatar/Nalaikh, 2026

GERMAN - MONGOLIAN INSTITUTE FOR RESOURCES AND TECHNOLOGY

Statutory Declaration

Shijee, Munkhsuld	B2200388
Last Name, First Name	Student ID Number

I hereby affirm in lieu of an oath that I provided the submitted bachelor thesis

Development of a Control System for a Street-Cleaning Robot

I did not use any sources other than those stated. In case that the work is additionally submitted on a data medium, I declare that the written and the electronic form are completely identical. The work was not submitted in the same or similar form to any examination authority.

Ulaanbaatar,Mongolia,May,27,2026

Place, Date



Signature

GERMAN - MONGOLIAN INSTITUTE FOR RESOURCES AND TECHNOLOGY

Acknowledgement

I would like to express my sincere appreciation to my supervisors, Ph.D. Odbileg N. and Prof. Ph.D., Ariunbolor P. for their continuous guidance, valuable technical advice, and constructive feedback throughout the course of this research. Their support and insightful suggestions played a significant role in shaping this work and enhancing its overall quality.

I would also like to extend my gratitude to my colleagues and fellow students for their encouragement, meaningful discussions, and support during the development of this project. Their contributions created a positive and motivating working environment.

Finally, I would like to thank my family for their patience, understanding, and unwavering support throughout my academic journey. Their encouragement has been a constant source of motivation in completing this thesis.

Abstract

This study presents the design and implementation of an autonomous mobile robot for street-cleaning applications, capable of environmental exploration and full-area coverage. The system is developed using the Pioneer P3DX mobile robot within the CoppeliaSim simulation environment and integrates ultrasonic sensors and a LiDAR sensor for perception and navigation. The primary objective of this research is to enable the robot to navigate along road curbs while acquiring information about the surrounding environment, and subsequently perform complete area coverage for cleaning tasks. During the exploration phase, the robot follows the curb using ultrasonic sensor feedback to maintain a consistent distance and avoid obstacles. This approach allows the robot to safely traverse the boundary of the environment and understand its spatial structure. Simultaneously, LiDAR sensor data is utilized to collect distance measurements of the environment, which are processed to obtain coordinate information representing the workspace. After completing the exploration phase, the acquired data is used to generate a coverage path, enabling the robot to systematically clean the entire area. External tools such as MATLAB can be employed to process the collected data, generate maps, and optimize the coverage path. Simulation results demonstrate that the robot can effectively follow road curbs, adapt to environmental variations, and collect consistent spatial data. Furthermore, the system is capable of performing structured coverage of the environment, ensuring efficient and complete cleaning. This work contributes to the development of intelligent autonomous robotic systems by integrating curb-following navigation, environmental data acquisition, and coverage planning into a unified framework. The proposed system can be extended with advanced mapping techniques, simultaneous localization and mapping (SLAM), and optimized path planning algorithms for real-world street-cleaning and urban maintenance applications.

GERMAN - MONGOLIAN INSTITUTE FOR RESOURCES AND TECHNOLOGY

Table of Contents

Abstract 3

1. Introduction 7

1.1 Background 7

1.2 Problem Statement 7

1.3 Objectives and Scope 8

2 Literature Review 9

2.1 Curb-Following and Boundary-Based Navigation 9

2.2 Sensor Technologies for Mobile Robots 9

2.3 Environmental Mapping and Data Processing 10

2.4 Coverage Path Planning 10

3 System Design 11

3.1 Hardware Components 11

3.1.1 Mobile Robot Platform 11

3.1.2 Control Unit – Arduino UNO 12

3.1.3 Motor Driver Module 13

3.1.4 Ultrasonic Sensors 14

3.1.5 LiDAR Sensor 14

3.1.6 Power Supply System 15

3.1.7 System Integration 16

3.1.8 Electronic Schematic Diagram 16

3.2 Software architecture 17

3.2.1 Embedded Control System (Arduino UNO) 17

3.2.2 Sensor Data Acquisition 18

3.2.3 Data Communication 18

3.2.4 MATLAB Integration 18

3.2.5 System Workflow 19

3.2.6 Control Algorithm Design 19

4 Methods 22
4.1 Simulation Environment Design 22
4.1.1 Environment Selection Criteria 22
4.1.2 Robot Platform 24
4.1.3 Ultrasonic Sensor Configuration 25
4.1.4 LiDAR Sensor Configuration 26
4.2 Integration of Robot and Sensor System 27
4.3 Exploration Method Selection 29
4.3.1 Exploration Behavior 29
4.4 Map Processing Strategy 30
4.4.1 Coordinate-Based Mapping 31
4.4.2 MATLAB-Based Map Processing 32
4.5 Cleaning Method Selection 33
5 Implementation 35
5.1 Robot Control and Sensor Implementation 35
5.1.1 Robot Model Configuration 37
5.1.2 Ultrasonic Sensor Implementation 39
5.1.3 Control Algorithm Implementation 41
5.2 LiDAR Implementation and Data Acquisition 42
5.3 LiDAR Data Extraction and MATLAB Processing 44
5.4 Mission Execution and System Evaluation 45
5.5 Cleaning Path Generation 46
6 Results and Discussion 47
6.1 Navigation Performance Analysis 48
6.2 LiDAR Data and Mapping Results 49
6.3 Cleaning Coverage Results 50
6.4 System Performance Evaluation 51
6.5 Discussion and Limitations 52

6.5.1 System Limitations 53
7 Conclusion and Future Work 54
7.1 Conclusion 54
7.2 Future Work 55
7.3 Summary 57
8 References 59
9 Appendices 62
9.1 Appendix A 62
9.2 Appendix B 64

1. Introduction

1.1 Background

In recent years, rapid urbanization and technological advancement have significantly increased the demand for automation in public infrastructure maintenance. Street cleaning and urban sanitation, in particular, remain labor-intensive processes that require substantial human effort, time, and operational cost. As cities continue to grow, maintaining clean and safe public environments has become increasingly challenging, creating a need for efficient and intelligent automated solutions.

Autonomous mobile robots have emerged as a promising approach to address these challenges. Such robots are capable of navigating environments, sensing surroundings, and performing tasks with minimal human intervention. In outdoor and semi-structured environments, such as urban streets, navigation becomes more complex due to varying terrain, obstacles, and environmental uncertainty. Therefore, reliable navigation strategies are essential for effective robot operation.

One practical navigation approach in urban environments is curb-following, where the robot moves along the boundary between the road and sidewalk. The curb serves as a natural and continuous reference, enabling the robot to maintain a consistent trajectory while covering relevant cleaning areas. By following this boundary, the robot can explore its environment systematically while ensuring that key regions are not missed.

To support navigation and perception, sensor integration plays a critical role. Ultrasonic sensors are widely used for short-range distance measurement and obstacle detection due to their simplicity and affordability. Meanwhile, LiDAR sensors provide more comprehensive environmental information by capturing distance measurements across a wide field of view. The combination of these sensors allows the robot to both follow curbs accurately and collect spatial data for further processing.

The collected environmental data can be transformed into coordinate-based representations, which are useful for mapping and coverage planning. These capabilities form the foundation for developing autonomous cleaning systems that can operate efficiently in real-world urban environments.

1.2 Problem Statement

Despite advancements in autonomous robotics, developing a reliable and efficient system for street-cleaning applications remains a challenging task. One of the primary difficulties lies in enabling a robot to navigate safely and consistently in semi-structured outdoor environments without relying on complex infrastructure or expensive positioning systems.

Traditional navigation approaches often require detailed pre-existing maps or advanced localization techniques such as GPS or SLAM, which may not always be reliable or feasible in urban conditions. Additionally, ensuring complete area coverage is a critical requirement for

cleaning applications, yet many existing systems struggle to guarantee that all regions are effectively covered without redundancy or omission.

Another challenge is the integration of multiple sensors to achieve both navigation and environmental perception. While ultrasonic sensors provide local obstacle information, they are limited in range and resolution. On the other hand, LiDAR sensors generate large amounts of data that must be processed efficiently to extract meaningful spatial information.

Therefore, there is a need for a simplified yet effective approach that enables a robot to:

- navigate using environmental features such as curbs,
- acquire spatial information without complex mapping systems,
- and perform complete area coverage for cleaning purposes.

1.3 Objectives and Scope

The main objective of this research is to design and implement an autonomous mobile robot capable of curb-following navigation, environmental data acquisition, and full-area coverage for cleaning applications.

The specific objectives of this study are as follows:

- To develop a curb-following control algorithm using ultrasonic sensor feedback for stable and reliable navigation.
- To integrate a LiDAR sensor for collecting environmental data during robot movement.
- To extract coordinate information (x, y) from sensor data for further processing and mapping.
- To generate coverage paths based on the collected environmental data using external tools such as MATLAB.
- To validate the performance of the proposed system through both simulation and real-world implementation.

The scope of this research includes both simulation-based and real-world experimentation. The system is initially developed and tested in the CoppeliaSim environment using the Pioneer P3DX mobile robot model to verify the control algorithms and sensor integration. Furthermore, the proposed approach is implemented in a real-world setup to evaluate its practical feasibility and performance under realistic conditions. This study focuses on curb-following navigation, environmental data acquisition, and coverage strategy.

2 Literature Review

Autonomous mobile robots (AMRs) are robotic systems capable of navigating and performing tasks in an environment without continuous human control. These robots rely on sensors, control algorithms, and decision-making processes to perceive their surroundings and respond accordingly. AMRs are widely used in applications such as industrial automation, warehouse logistics, environmental monitoring, and cleaning operations.

A key requirement for autonomous robots is the ability to navigate safely and efficiently. This involves obstacle detection, path planning, and motion control. In semi-structured environments, such as urban streets, robots must operate under varying conditions, making robust and adaptable navigation strategies essential.

2.1 Curb-Following and Boundary-Based Navigation

Boundary-based navigation is one of the simplest and most effective strategies for mobile robots operating in structured or semi-structured environments. Instead of relying on global positioning or complex mapping, the robot uses environmental features such as walls or curbs as references for navigation.

Curb-following is a specific type of boundary-based navigation where the robot moves along the edge of a road or sidewalk while maintaining a fixed distance from the curb. This method is particularly suitable for street-cleaning applications, as it ensures that the robot systematically covers the edges of the environment where debris is commonly accumulated.

Various control strategies have been proposed for wall and boundary following, including proportional (P), proportional-integral (PI), and proportional-integral-derivative (PID) controllers. These approaches use sensor feedback to minimize the error between the desired and measured distance from the boundary. Among them, proportional control is often sufficient for stable and real-time performance in simple navigation tasks.

2.2 Sensor Technologies for Mobile Robots

Sensors play a critical role in enabling autonomous robots to perceive their environment. Two commonly used sensors in mobile robotics are ultrasonic sensors and LiDAR sensors.

Ultrasonic sensors measure distance by emitting sound waves and calculating the time taken for the reflected signal to return. They are widely used for short-range obstacle detection and boundary following due to their low cost and simplicity. However, their accuracy can be affected by surface properties and environmental noise.

LiDAR (Light Detection and Ranging) sensors provide more detailed environmental information by emitting laser beams and measuring the reflected signals. LiDAR systems can generate high-resolution distance data over a wide field of view, making them suitable for mapping and spatial analysis. Compared to ultrasonic sensors, LiDAR offers higher accuracy and reliability but at a higher computational and hardware cost.

The combination of ultrasonic and LiDAR sensors enables a balance between real-time control and detailed environmental perception.

2.3 Environmental Mapping and Data Processing

Mapping is a fundamental component of autonomous navigation, allowing robots to represent and understand their environment. Sensor data, particularly from LiDAR, can be converted into coordinate-based representations such as point clouds or grid maps.

Point cloud data consists of a collection of spatial points that represent the structure of the environment. This data can be processed to identify obstacles, boundaries, and free space. In many cases, external tools such as MATLAB are used to analyze and visualize this data, enabling further processing such as map generation and filtering.

More advanced mapping approaches, such as Simultaneous Localization and Mapping (SLAM), allow robots to build maps while estimating their own position. However, these methods are computationally intensive and may not be necessary for simpler applications where boundary-based navigation is sufficient.

2.4 Coverage Path Planning

Coverage path planning is the process of determining a path that allows a robot to traverse an entire area without leaving gaps or unnecessary overlap. This is particularly important in cleaning applications, where complete coverage directly impacts performance.

Common coverage strategies include zigzag (boustrophedon) patterns, spiral paths, and decomposition-based methods. These approaches divide the environment into manageable sections and generate paths that ensure systematic coverage.

In practical implementations, coverage paths are often generated based on environmental data obtained from sensors. By using coordinate information extracted from LiDAR data, it is possible to design efficient paths that adapt to the shape and size of the environment.

3 System Design

The proposed system is an autonomous mobile robot designed for street-cleaning applications. The robot is capable of navigating along road curbs, collecting environmental data, and performing full-area coverage based on the acquired information. The system integrates control algorithms, multiple sensors, and data processing techniques to achieve autonomous operation.

The overall workflow of the system consists of three main stages: environmental exploration, data acquisition, and coverage execution. During the exploration phase, the robot follows the curb using ultrasonic sensor feedback while simultaneously collecting spatial data using a LiDAR sensor. The acquired data is then processed to obtain coordinate information representing the environment. Finally, a coverage path is generated, allowing the robot to systematically clean the entire area.

3.1 Hardware Components

The hardware configuration of the proposed system consists of sensing units, a control unit, actuation components, and a mobile platform. These components work together to enable real-time navigation, environmental perception, and cleaning operation.

3.1.1 Mobile Robot Platform

The robot platform used in this study is based on a differential-drive mechanism, which consists of two independently driven wheels mounted on either side of the chassis and a passive caster wheel for stability. This configuration enables the robot to achieve various types of motion, including forward movement, backward movement, in-place rotation, and smooth curved trajectories by adjusting the relative velocities of the left and right wheels.

From a kinematic perspective, the motion of the robot is governed by the angular velocities of the two drive wheels. When both wheels rotate at equal speeds, the robot moves in a straight line. When there is a difference in wheel speeds, the robot follows a circular path, where the radius of curvature depends on the velocity difference. If the wheels rotate in opposite directions, the robot rotates about its central axis. This simple yet effective motion model makes the differential-drive system highly suitable for mobile robotic applications.

One of the key advantages of the differential-drive platform is its mechanical simplicity. The absence of complex steering mechanisms reduces the number of moving parts, leading to lower manufacturing cost and easier maintenance. This simplicity also translates into straightforward control implementation, as the robot's motion can be directly controlled through wheel velocities without requiring complex kinematic transformations.

Another important advantage is its high maneuverability, particularly in confined environments such as sidewalks, road curbs, and narrow pathways. The ability to perform zero-radius turns allows the robot to navigate tight corners and avoid obstacles efficiently. This characteristic is especially beneficial for cleaning applications where the robot must operate in structured but constrained environments.

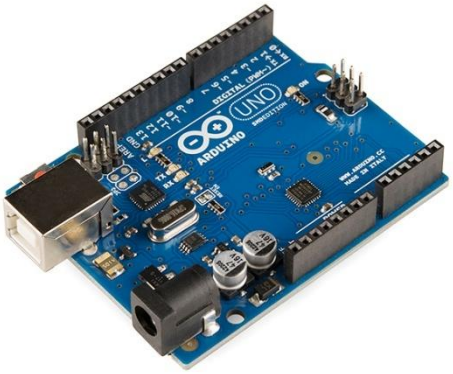
The robot chassis serves as the structural foundation for all system components. It is designed to provide stable mounting for sensors, electronic control units, power supply modules, and actuators. The placement of components is carefully considered to maintain balance and ensure uniform weight distribution, which improves stability during motion.

In this implementation, the platform accommodates ultrasonic sensors for real-time navigation and a LiDAR sensor for environmental mapping. These sensors are mounted at specific positions to maximize field of view and minimize occlusion. The control hardware, including the microcontroller and motor driver, is centrally located to reduce wiring complexity and improve signal reliability.

The power system, typically consisting of rechargeable batteries, is integrated into the chassis to provide sufficient energy for both computation and actuation. Proper power distribution ensures consistent motor performance and stable sensor operation throughout the robot's operation.

3.1.2 Control Unit – Arduino UNO

The Arduino UNO microcontroller serves as the central processing unit of the robot. It is responsible for executing control algorithms, processing sensor data, and generating control signals for motor actuation.



Key features of the Arduino UNO include:

- ATmega328P microcontroller
- 16 MHz clock speed
- Digital and analog input/output pins
- PWM (Pulse Width Modulation) capability

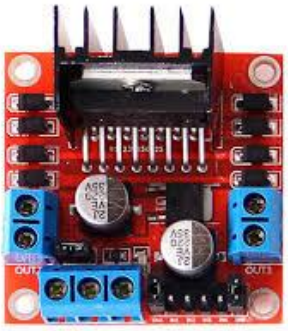
The Arduino performs the following functions:

- Reads distance measurements from ultrasonic sensors
- Calculates the error between desired and measured distances
- Executes the curb-following control algorithm
- Generates PWM signals to control motor speed
- Controls motor direction through digital output pins

Due to its low cost, ease of programming, and real-time performance, the Arduino UNO is well-suited for embedded robotic applications.

3.1.3 Motor Driver Module

The motor driver acts as an interface between the Arduino and the DC motors. Since the Arduino cannot supply sufficient current to drive motors directly, the motor driver amplifies control signals and provides the necessary power.



In this system, a dual H-bridge motor driver (L298N) is used. The motor driver enables:

- Bidirectional motor control (forward and reverse)
- Speed control using PWM signals
- Independent control of left and right motors

Each motor is controlled using two digital inputs (for direction) and one PWM input (for speed). This allows precise control of robot movement, including turning and speed adjustment during curb-following.

3.1.4 Ultrasonic Sensors

Ultrasonic sensors are used for short-range distance measurement and play a critical role in curb-following navigation and obstacle avoidance.



The sensors operate by:

1. Emitting an ultrasonic pulse
2. Receiving the reflected signal
3. Calculating distance based on time-of-flight

In this system, multiple ultrasonic sensors are positioned strategically:

- Front sensor: Detects obstacles directly ahead
- Front-right sensor: Assists in detecting angled surfaces
- Right sensor: Measures distance from the curb

These sensors provide real-time feedback that is used by the control algorithm to maintain a desired distance from the curb.

Typical characteristics of ultrasonic sensors include:

- Measurement range: approximately 2 cm to 400 cm
- Moderate accuracy
- Sensitivity to surface angle and material

Despite their limitations, ultrasonic sensors are effective for simple and fast distance measurement in real-time control systems.

3.1.5 LiDAR Sensor

The LiDAR (Light Detection and Ranging) sensor is used for environmental perception and data acquisition. It provides high-resolution distance measurements over a wide scanning angle.

The LiDAR operates by emitting laser beams and measuring the time taken for the reflected signal to return. This allows the sensor to calculate distances to surrounding objects.



Key characteristics:

- Wide field of view (360° depending on model)
- High measurement accuracy
- Ability to generate dense spatial data

In this system, the LiDAR is used to:

- Scan the environment during robot motion
- Collect distance data points
- Convert measurements into coordinate form (x, y)

The resulting coordinate data is used for mapping and coverage path planning in MATLAB.

3.1.6 Power Supply System

The robot is powered by a rechargeable battery system, which supplies energy to all components, including the Arduino, motor driver, sensors, and motors.

The power system must meet the following requirements:

- Provide sufficient voltage and current for motors
- Ensure stable operation of the microcontroller
- Prevent voltage drops during high load conditions

Voltage regulation may be required to supply appropriate levels to different components (e.g., 5V for Arduino and sensors, higher voltage for motors).

3.1.7 System Integration

All hardware components are integrated to form a complete robotic system. The ultrasonic sensors and LiDAR provide input data, the Arduino processes this data, and the motor driver executes movement commands.

The integration ensures:

- Real-time response to environmental changes
- Stable navigation along the curb
- Continuous data acquisition for mapping

This combination of hardware components enables the robot to perform autonomous navigation and cleaning tasks effectively.

3.1.8 Electronic Schematic Diagram

The complete electronic schematic diagram of the proposed robot system is shown in Figure 3.X. This diagram illustrates the electrical connections between the main components, including the Arduino UNO, motor driver, ultrasonic sensors, LiDAR sensor, and power supply system.

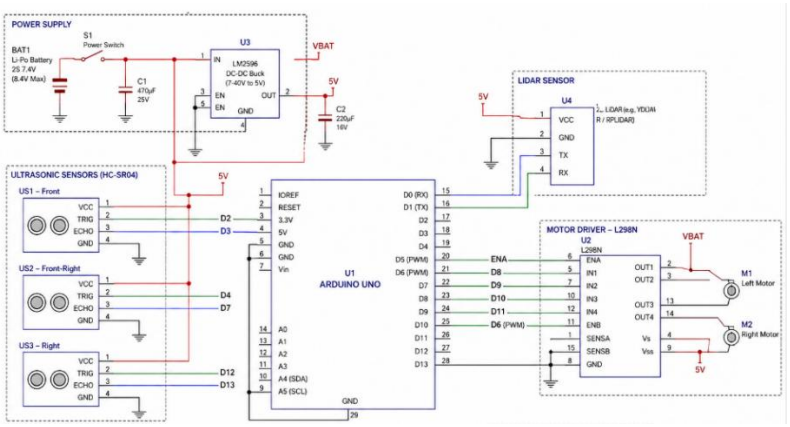
In this system, the Arduino UNO acts as the central controller and interfaces with all sensors and actuators. The ultrasonic sensors are connected to the digital input/output pins of the Arduino, where each sensor uses separate trigger and echo pins for distance measurement. These sensors provide real-time data for curb-following and obstacle detection.

The motor driver (L298N) is connected to the Arduino through digital control pins and PWM pins. The Arduino sends control signals to the motor driver to regulate the speed and direction of the two DC motors. The motor driver is powered directly by the battery, allowing it to supply sufficient current to the motors.

The LiDAR sensor is connected via serial communication (TX/RX) to the Arduino or an external processing unit. It continuously scans the environment and provides distance measurements, which are later converted into spatial coordinates for mapping and coverage planning.

The power supply system consists of a battery that provides voltage to both the motor driver and the Arduino. A voltage regulation mechanism ensures stable 5V supply for the microcontroller and sensors. All components share a common ground (GND) to ensure proper signal reference and stable operation.

This schematic provides a complete overview of the electrical integration of the system and ensures proper communication between sensing, control, and actuation components.



3.2 Software architecture

The software architecture of the proposed autonomous robot system is designed to integrate real-time control, sensor data acquisition, and external data processing into a unified framework. The system is structured in a way that separates time-critical embedded control from computationally intensive data processing tasks. This separation improves system stability, simplifies implementation, and allows each component to operate efficiently within its own domain.

The overall software system is composed of three main parts: the embedded control system implemented on the Arduino UNO, the sensor data acquisition layer, and the external processing environment implemented using MATLAB. These components work together to enable autonomous navigation, environmental perception, and coverage path generation.

3.2.1 Embedded Control System (Arduino UNO)

The Arduino UNO acts as the central embedded controller responsible for real-time operation of the robot. It continuously reads data from the ultrasonic sensors and generates control signals for the motor driver. The control logic is implemented using embedded C/C++ within the Arduino development environment.

The control system operates in a continuous loop, where sensor data is acquired, processed, and used to determine motor commands. This loop allows the robot to respond immediately to environmental changes. The primary task of the Arduino is to maintain a consistent distance from the curb while avoiding obstacles.

Distance measurements from the ultrasonic sensors are used to calculate the deviation from the desired path. Based on this deviation, the control system adjusts the speed of the left and right motors. This adjustment enables the robot to correct its trajectory and maintain stable motion along the curb.

The Arduino also handles obstacle detection. When an obstacle is detected within a predefined distance, the control system temporarily overrides the normal navigation behavior and executes a turning maneuver to avoid collision. Once the obstacle is cleared, the robot resumes its normal curb-following operation.

3.2.2 Sensor Data Acquisition

The system utilizes both ultrasonic sensors and a LiDAR sensor to gather environmental information. Each sensor type serves a different purpose within the system.

Ultrasonic sensors are used for short-range measurements and are directly connected to the Arduino. These sensors provide immediate feedback required for real-time control. Their data is processed continuously to ensure accurate navigation and obstacle avoidance.

The LiDAR sensor is used for long-range environmental scanning. It continuously collects distance measurements over a wide field of view, providing a detailed representation of the surroundings. Unlike ultrasonic sensor data, LiDAR data is not used directly for real-time control but is instead collected for mapping and further analysis.

The data from the LiDAR sensor is processed to extract spatial coordinates, which represent the structure of the environment. This data forms the basis for mapping and coverage planning.

3.2.3 Data Communication

The communication between system components is achieved through a combination of direct hardware connections and data transfer mechanisms. Ultrasonic sensors communicate directly with the Arduino through digital input/output pins. The LiDAR sensor communicates through a serial interface, allowing continuous transmission of measurement data.

In the simulation environment, data communication is implemented using signal-based mechanisms. Sensor data is packaged into structured formats and transmitted between different modules. This approach ensures efficient data exchange and synchronization between sensing and processing components.

The communication system is designed to minimize delays and ensure that all data required for control and analysis is available when needed.

3.2.4 MATLAB Integration

MATLAB is used as an external processing tool to analyze the data collected from the LiDAR sensor. The extracted coordinate data is imported into MATLAB, where it is processed and visualized.

The processing steps include generating a map of the environment using coordinate data, identifying the boundaries of the workspace, and creating a coverage path that ensures the entire area is traversed. MATLAB provides advanced computational capabilities that are not available on the Arduino, making it suitable for handling large datasets and complex calculations.

The integration of MATLAB allows the system to extend beyond simple navigation and perform higher-level tasks such as mapping and path planning.

3.2.5 System Workflow

The software system operates through a coordinated workflow involving all components. The Arduino continuously controls the robot's movement based on ultrasonic sensor data. At the same time, the LiDAR sensor collects environmental data during robot motion. This data is transmitted and stored for processing.

Once sufficient data has been collected, it is processed in MATLAB to generate a map and coverage path. The robot can then use this information to perform systematic cleaning of the entire area.

3.2.6 Control Algorithm Design

The control algorithm of the proposed autonomous robot system is designed to achieve stable curb-following navigation while ensuring safe obstacle avoidance. The algorithm is implemented on the Arduino UNO and operates in real time using feedback from ultrasonic sensors.

The control strategy is based on a proportional (P) control approach combined with rule-based decision logic for obstacle avoidance. This hybrid approach allows the robot to maintain a desired distance from the curb while reacting immediately to environmental changes.

Distance Measurement and Error Calculation

The primary input to the control algorithm is the distance measured by the right-side ultrasonic sensor. This sensor provides the distance between the robot and the curb.

Let the desired distance from the curb be denoted as (d_{ref}), and the measured distance be (d_{meas}). The control error is defined as:

e = d_{ref} - d_{meas}

This error represents how far the robot deviates from the desired path. A positive error indicates that the robot is too far from the curb, while a negative error indicates that the robot is too close.

Proportional Control Law

To correct the deviation, a proportional control law is applied. The control signal (u) is calculated as:

u = Kp \cdot e

where (Kp) is the proportional gain that determines the responsiveness of the system.

The control signal is then used to adjust the velocities of the left and right motors.

Differential Drive Control

The robot uses a differential drive mechanism, where the left and right wheel velocities are controlled independently. The motor velocities are computed as:

vL = v_{base} - u

vR = v_{base} + u

where (v_{base}) is the nominal forward velocity of the robot.

This control structure allows the robot to steer toward or away from the curb depending on the sign of the error. If the robot is too far from the curb, it turns toward it. If it is too close, it turns away.

Obstacle Detection and Avoidance

In addition to curb-following, the control algorithm includes an obstacle avoidance mechanism based on front-facing ultrasonic sensors.

The algorithm continuously monitors the distance measured by the front and front-right sensors. If an obstacle is detected within a predefined threshold distance (d_{th}), the robot overrides the normal control behavior and executes a turning maneuver.

The decision logic is defined as follows:

- If (d_{front} < d_{th}), the robot turns left
- If (d_{front-right} < d_{th}), the robot adjusts its trajectory
- If no obstacle is detected, the robot continues curb-following

This ensures safe navigation in dynamic environments.

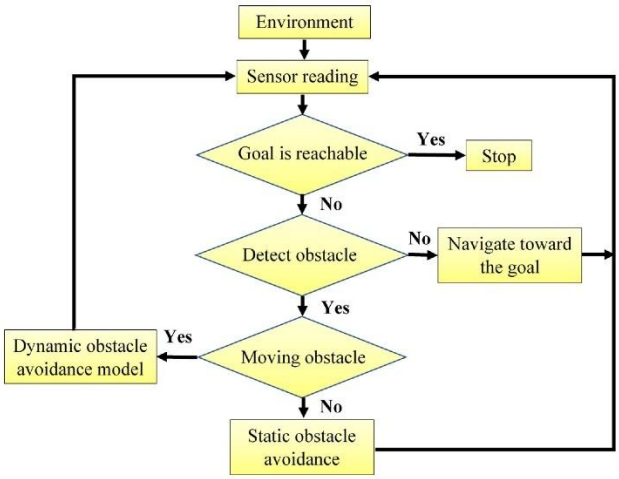
Wall Loss and Recovery Strategy

A special condition occurs when the right-side ultrasonic sensor does not detect any object. This indicates that the robot has lost the curb.

In this case, the algorithm switches to a recovery mode. The robot adjusts its trajectory by turning toward the expected direction of the curb until the sensor detects a valid distance again. This prevents the robot from drifting away from the boundary.

Control Loop Execution

The entire control algorithm operates within a continuous loop on the Arduino. The loop performs the following sequence of operations:



1. Read sensor data from ultrasonic sensors
2. Check for obstacle conditions
3. Calculate distance error
4. Compute control signal using proportional control
5. Adjust motor velocities
6. Repeat the process

This loop runs at high frequency, allowing the robot to react quickly to changes in the environment.

Stability and Performance Considerations

The proportional gain plays a critical role in system performance. If is too small, the robot responds slowly and may drift from the desired path. If is too large, the system may become unstable and oscillate.

Through experimentation, an appropriate value of is selected to balance responsiveness and stability. The chosen control strategy provides sufficient performance for curb-following without requiring more complex control methods such as PID.

The control algorithm combines proportional control with rule-based decision logic to achieve stable and reliable navigation. The use of ultrasonic sensors for real-time feedback and differential drive control enables the robot to follow the curb accurately while avoiding obstacles. This approach provides a simple yet effective solution for autonomous navigation in semi-structured environments.

4 Methods

4.1 Simulation Environment Design

4.1.1 Environment Selection Criteria

The selection of the simulation environment is a critical step in evaluating the performance of the proposed autonomous cleaning robot system. The environment must be sufficiently realistic to represent actual operating conditions, while also being structured enough to allow controlled testing and analysis.

In this study, a semi-structured environment is selected to represent typical roadside or sidewalk conditions where curb-following navigation is applicable. The environment is designed to include continuous boundaries that simulate road curbs, along with open areas and isolated obstacles. This type of layout reflects practical scenarios where a cleaning robot is expected to operate, such as pedestrian walkways, road edges, and public outdoor spaces.

The decision to focus on a semi-structured environment is based on the balance between complexity and controllability. Highly cluttered environments with numerous small objects were not selected, as they introduce excessive variability and require more advanced perception and planning algorithms. Instead, the environment is composed of fixed structures that define clear navigation boundaries while still presenting challenges such as curved paths and obstacle avoidance.

Another important consideration is the accessibility of the environment. The selected layout ensures that the robot can traverse most of the area without being physically restricted by narrow passages or complex geometries. This is important because the objective of the study is to evaluate navigation, mapping, and coverage performance rather than mechanical limitations of the robot.

The environment also includes geometric variations such as straight segments and curved boundaries. These variations are intentionally introduced to test the robot's ability to maintain stable navigation under different conditions. Straight boundaries allow evaluation of steady-state control performance, while curved sections test the robot's responsiveness and turning capability.

Additionally, the inclusion of a central obstacle provides a realistic challenge for both sensing and navigation. The obstacle requires the robot to adapt its trajectory while continuing to follow the boundary and collect environmental data. This ensures that the system is evaluated under conditions that resemble real-world scenarios rather than idealized test cases.

4.1.2 Robot Platform

The robotic platform employed in this study is a differential-drive mobile robot modeled within the CoppeliaSim simulation environment. The selection of this platform is based on its widespread use in mobile robotics, its mechanical simplicity, and its suitability for real-time control applications. The robot is designed to operate on a planar surface and is capable of performing navigation tasks such as straight-line motion, turning, and boundary following.

The mechanical structure of the robot consists of two independently driven wheels positioned on either side of the chassis and a passive caster wheel that provides balance and support. This configuration ensures that the robot maintains stability during motion while minimizing mechanical complexity. The absence of additional steering mechanisms reduces both control complexity and computational requirements, making it well-suited for embedded control systems such as the Arduino UNO used in this study.

The motion of the robot is governed by differential-drive kinematics. In this configuration, the trajectory of the robot is determined by the relative velocities of the left and right wheels. When both wheels rotate at identical speeds, the robot moves forward in a straight line. When the wheel speeds differ, the robot follows a curved trajectory whose radius depends on the velocity difference. If the wheels rotate in opposite directions, the robot is capable of rotating about its central axis without translating. This ability to perform in-place rotation is particularly useful when the robot encounters obstacles or needs to reorient itself within confined spaces.

From a control perspective, the differential-drive mechanism allows direct mapping between control signals and robot motion. Instead of computing complex steering angles, the control system simply adjusts the speed of each wheel. This makes the system highly responsive and suitable for real-time implementation, where quick adjustments are required based on sensor feedback.

The robot operates strictly within a two-dimensional plane, as the application is focused on ground-based navigation and cleaning. Although the simulation environment is inherently three-dimensional, vertical dynamics such as elevation changes or tilting are not considered. This simplification allows the control system to focus exclusively on planar motion, reducing both computational load and modeling complexity.

Another important aspect of the robot platform is its geometric structure. The robot body has a finite width and length, which must be considered during navigation, especially when moving near boundaries or obstacles. The physical dimensions of the robot influence how closely it can approach the curb and how it behaves in tight spaces. During curved motion, the outer and inner wheels travel different distances, which can lead to slight deviations from the ideal path. These effects are inherent to differential-drive systems and are taken into account during control design.

The placement of sensors on the robot is also closely related to the platform design. The robot provides suitable mounting locations for ultrasonic sensors and the LiDAR sensor. The ultrasonic sensors are positioned to capture forward and lateral distance measurements, while the LiDAR is mounted at an elevated position to maximize its field of view. The structural design

of the robot ensures that these sensors operate without obstruction, allowing accurate data acquisition.

In addition, the robot platform is selected to ensure compatibility with both simulation and real-world implementation. The behavior of the simulated robot closely approximates that of a physical differential-drive robot, allowing the control algorithms developed in simulation to be transferred to hardware with minimal modification. This consistency is essential for validating the practicality of the proposed system.

The dynamic behavior of the robot is influenced by factors such as wheel friction, inertia, and contact with the ground surface. In the simulation environment, these parameters are configured to approximate realistic conditions. Proper tuning of these parameters ensures that the robot exhibits stable motion and does not experience unrealistic slipping or oscillations.

4.1.3 Ultrasonic Sensor Configuration

Ultrasonic sensors are employed in this study as the primary means of real-time distance measurement for navigation and obstacle avoidance. These sensors are selected due to their simplicity, low cost, and ability to provide reliable short-range distance measurements in real-time applications. Within the context of the proposed system, ultrasonic sensors play a critical role in enabling reactive behavior, particularly for curb-following and collision prevention.

The operating principle of an ultrasonic sensor is based on the time-of-flight of sound waves. The sensor emits a high-frequency ultrasonic pulse, which propagates through the air until it encounters an object. The pulse is then reflected back to the sensor, and the time taken for this round-trip is measured. Using the known speed of sound, the distance to the object can be calculated. This method allows the sensor to detect objects without physical contact and provides continuous distance feedback during robot operation.

In the proposed system, three ultrasonic sensors are integrated into the robot and positioned strategically to cover critical directions. The first sensor is mounted at the front of the robot, facing directly forward. This sensor is responsible for detecting obstacles in the robot's immediate path and plays a key role in preventing frontal collisions. The second sensor is positioned at a front-right angle, allowing it to detect objects that are not directly in front but still within the robot's forward trajectory. This angled placement enhances the robot's perception and provides additional information for smoother navigation around obstacles.

The third sensor is mounted on the right side of the robot and is dedicated to curb-following. This sensor continuously measures the distance between the robot and the boundary of the environment. Its placement is crucial, as it directly influences the accuracy of the control algorithm. By providing consistent lateral distance measurements, the sensor enables the robot to maintain a stable and controlled trajectory along the curb.

The configuration of these sensors is carefully designed to ensure complementary coverage. The front and front-right sensors handle obstacle detection and avoidance, while the right-side sensor supports boundary tracking. This division of functionality allows the system to operate efficiently without requiring complex sensor fusion algorithms.

The performance of ultrasonic sensors is influenced by several factors, including detection range, beam angle, and environmental conditions. In the simulation environment, these parameters are configured to approximate realistic behavior. The detection range is set to ensure that the robot can detect obstacles with sufficient time to react, while the beam angle determines the spread of the sensor's detection field. A wider beam angle increases coverage but may reduce precision, whereas a narrower beam provides more accurate measurements but limited coverage.

One limitation of ultrasonic sensors is their sensitivity to surface properties and angle of incidence. Objects with irregular shapes or soft surfaces may produce weak or inconsistent reflections. Additionally, measurements can become less reliable at larger distances due to signal attenuation. However, within the operating range required for curb-following and obstacle avoidance, these limitations are manageable and do not significantly impact system performance.

Another important consideration is the update rate of the sensors. The sensors must provide distance measurements at a frequency that is sufficient for real-time control. If the update rate is too low, the robot may not respond quickly enough to changes in the environment. In this study, the sensor update rate is configured to ensure smooth and responsive navigation.

The integration of ultrasonic sensors with the control system is designed to be straightforward. The sensor readings are directly used as inputs to the control algorithm without requiring complex preprocessing. This allows the system to maintain low computational overhead while still achieving effective navigation behavior.

4.1.4 LiDAR Sensor Configuration

In addition to ultrasonic sensors, a LiDAR sensor is integrated into the robotic system to provide comprehensive environmental perception. While ultrasonic sensors are effective for short-range, direction-specific measurements, they are limited in spatial coverage and resolution. The LiDAR sensor complements these limitations by offering wide-angle scanning and high-density distance measurements, enabling the robot to capture a detailed representation of its surroundings.

The LiDAR operates based on the principle of laser time-of-flight. It emits laser beams in multiple directions and measures the time taken for each beam to reflect back from surrounding objects. Using this time measurement and the speed of light, the sensor calculates the distance to objects with high accuracy. This process is repeated across a range of angles, producing a set of distance measurements that correspond to different directions around the robot.

In this study, the LiDAR is configured to operate in a two-dimensional scanning mode. This means that all measurements are taken within a single horizontal plane, which is sufficient for ground-level navigation and mapping tasks. The sensor generates a scan consisting of multiple points distributed over its angular field of view. Each point represents the distance to the nearest object in a particular direction.

The LiDAR sensor is mounted on the top of the robot to ensure an unobstructed field of view. This elevated position minimizes occlusion caused by the robot's body and allows the sensor to capture data from all directions within its scanning range. Proper mounting is essential for

accurate data acquisition, as any obstruction or misalignment can result in incomplete or distorted measurements.

The scanning parameters of the LiDAR, including maximum range, angular resolution, and scanning frequency, are configured to match the requirements of the system. The maximum range determines how far the sensor can detect objects, while the angular resolution defines the number of measurement points per scan. A higher resolution provides more detailed information but increases computational load. The scanning frequency determines how often the sensor updates its measurements, which is important for maintaining consistency with the robot's motion.

The data produced by the LiDAR is initially expressed in polar coordinates, where each measurement consists of a distance and an associated angle. To make this data suitable for mapping and analysis, it is converted into Cartesian coordinates. This transformation allows each point to be represented in terms of its position relative to the robot, forming a spatial representation of the environment.

Unlike ultrasonic sensors, which are used directly for real-time control, the LiDAR data is primarily used for data collection and post-processing. The sensor continuously scans the environment as the robot moves, and the collected data is stored for later use. This separation of roles allows the system to maintain fast and responsive control using ultrasonic sensors while still benefiting from the detailed environmental information provided by the LiDAR.

One of the key advantages of using LiDAR is its ability to capture complex geometric features, such as curved boundaries and irregularly shaped obstacles. In the simulation environment, this is evident in the representation of circular objects and smooth boundary curves. The LiDAR provides sufficient resolution to distinguish these features, which is essential for accurate mapping and coverage planning.

However, LiDAR sensors also have limitations. Measurements may be affected by reflective surfaces, sharp angles, or partial occlusions. Additionally, the accuracy of the data depends on the calibration and alignment of the sensor. In the simulation environment, these factors are controlled to ensure consistent performance, but they must be considered in real-world applications.

The integration of the LiDAR with the overall system is designed to be non-intrusive. The sensor operates independently of the control loop, continuously generating data without interfering with navigation. The collected data is transmitted and stored using appropriate data handling methods, allowing it to be processed later in MATLAB.

4.2 Integration of Robot and Sensor System

The integration of the robot platform with its sensing components is a critical aspect of the overall system design. In this study, the integration is carefully structured to ensure that all sensors and actuators operate in a coordinated manner while maintaining clear separation between real-time control and data acquisition processes.

At the core of the system, the robot platform acts as the physical base that supports both motion and sensing. The ultrasonic sensors and LiDAR sensor are mounted on the robot in positions

that maximize their effectiveness while minimizing interference. The placement of each sensor is determined based on its functional role within the system. The ultrasonic sensors are positioned to provide immediate distance feedback for navigation, while the LiDAR sensor is placed at an elevated position to capture a broader view of the environment.

From a system architecture perspective, the integration is designed around two parallel operational layers. The first layer is the real-time control layer, which uses data from ultrasonic sensors to guide the robot's movement. This layer is responsible for maintaining curb-following behavior and avoiding obstacles. The second layer is the perception and data acquisition layer, which uses the LiDAR sensor to collect environmental data for mapping and analysis.

This separation of responsibilities is intentional and plays an important role in system efficiency. By isolating real-time control from data processing, the system avoids computational overload and ensures that navigation remains responsive. The ultrasonic sensors provide fast and lightweight measurements that can be processed directly by the control algorithm, while the LiDAR handles more complex data collection tasks without affecting immediate robot behavior.

The integration also involves synchronization between sensing and motion. As the robot moves, both ultrasonic sensors and the LiDAR continuously generate data. The control system must process ultrasonic data at a sufficiently high rate to ensure stable navigation, while LiDAR data is collected over time and accumulated to form a representation of the environment. This requires careful timing and coordination to ensure that sensor readings correspond accurately to the robot's position.

Another important aspect of integration is the transformation of sensor data into a consistent reference frame. The LiDAR measurements are initially obtained relative to the sensor's local coordinate system. These measurements must be transformed into the robot's coordinate frame and, if necessary, into a global reference frame. This transformation ensures that all data points are aligned and can be used effectively for mapping and analysis.

The communication between different components of the system is handled through defined data exchange mechanisms. In the simulation environment, signals are used to transfer data between scripts, allowing the LiDAR data to be accessed by other parts of the system. In a physical implementation, similar communication would be achieved using serial data transmission between the robot and an external processing unit.

The integration process also considers system robustness and reliability. Each sensor operates independently, which means that a failure or limitation in one sensor does not completely compromise the system. For example, even if LiDAR data is temporarily unavailable, the robot can still navigate using ultrasonic sensors. This redundancy improves the overall reliability of the system.

Furthermore, the integration is designed to be scalable. Additional sensors or processing modules can be incorporated without fundamentally changing the system architecture. This flexibility is important for future improvements, such as incorporating advanced localization or mapping techniques.

4.3 Exploration Method Selection

The selection of an appropriate exploration method is a crucial step in the development of the autonomous robot system, as it directly influences the efficiency of data collection and the quality of the resulting map. The chosen method must satisfy several requirements, including simplicity of implementation, reliability in unknown environments, and compatibility with the available sensing and control capabilities of the robot.

In this study, multiple exploration strategies were considered before selecting the final approach. One of the simplest methods is random exploration, where the robot moves without a predefined direction and relies solely on obstacle avoidance to navigate. While this method is easy to implement, it lacks efficiency and does not guarantee complete coverage of the environment. The robot may repeatedly visit the same areas while leaving other regions unexplored, resulting in incomplete or uneven data collection.

Another approach involves map-based exploration techniques, such as those based on simultaneous localization and mapping (SLAM). These methods allow the robot to build a map of the environment while simultaneously determining its position within that map. Although highly effective, SLAM-based approaches require significant computational resources and complex algorithms. Given the scope of this study and the goal of maintaining a lightweight and practical system, such methods were not selected.

Instead, a curb-following strategy is adopted as the primary exploration method. This approach leverages the natural structure of the environment, using boundaries such as curbs or walls as a guide for navigation. By following a continuous boundary, the robot can systematically traverse the perimeter of the environment while maintaining a consistent reference.

The decision to use curb-following is based on several key considerations. First, it provides a deterministic exploration path, ensuring that the robot covers the entire boundary without unnecessary repetition. This is particularly important for mapping, as the boundary defines the overall shape of the environment. Second, the method relies only on local sensor information, specifically distance measurements from ultrasonic sensors, which makes it suitable for real-time implementation on a microcontroller.

Another advantage of curb-following is its robustness in semi-structured environments. Even in the presence of minor irregularities or obstacles, the robot can continue to follow the boundary by adjusting its trajectory. This makes the method adaptable to real-world conditions where perfect geometric structures are not always present.

However, the selection of curb-following also involves certain trade-offs. Since the method focuses on boundary traversal, it does not inherently guarantee full exploration of interior regions. This limitation is addressed in later stages of the system through coverage path planning, which ensures that the entire area is eventually traversed.

4.3.1 Exploration Behavior

Once the exploration method is defined, it is necessary to describe how the robot behaves during execution. The exploration behavior in this study is based on continuous curb-following combined with reactive adjustments to environmental conditions. This behavior allows the robot

to move along the boundary of the environment while simultaneously adapting to changes such as curvature and obstacles.

At the start of the exploration phase, the robot begins moving forward while monitoring the distance to the curb using the right-side ultrasonic sensor. The control system continuously adjusts the robot's trajectory to maintain a predefined distance from the boundary. This results in smooth motion along the curb, where the robot effectively "tracks" the edge of the environment.

As the robot progresses, it encounters different types of boundary geometries. In straight segments, the robot maintains a relatively constant heading, and only minor corrections are required to keep the desired distance. In curved sections, the robot must continuously adjust its direction to follow the shape of the boundary. This is achieved by dynamically modifying the relative speeds of the left and right wheels, allowing the robot to perform gradual turns without abrupt changes in motion.

In addition to boundary tracking, the robot must handle situations where obstacles are present along its path. These obstacles may appear near the boundary or directly in front of the robot. The front and front-right ultrasonic sensors are used to detect such obstacles. When an obstacle is detected within a predefined threshold distance, the robot temporarily alters its behavior to avoid collision.

During obstacle avoidance, the robot deviates from its boundary-following path and performs a turning maneuver. The direction and magnitude of this maneuver depend on the location of the detected obstacle. Once the obstacle is no longer detected, the robot gradually returns to its original path and resumes curb-following behavior. This transition is designed to be smooth to prevent instability or sudden changes in motion.

Another important aspect of exploration behavior is handling situations where the boundary is temporarily lost. This can occur if the sensor fails to detect the curb due to gaps, sharp corners, or measurement limitations. In such cases, the robot enters a recovery mode, where it adjusts its trajectory to search for the boundary. Once the boundary is detected again, normal operation resumes.

Throughout the exploration process, the robot maintains continuous forward motion, ensuring that the environment is traversed efficiently. The behavior is inherently reactive, meaning that all decisions are based on current sensor readings rather than precomputed paths. This allows the robot to operate effectively in unknown environments without requiring prior knowledge.

Simultaneously, the LiDAR sensor operates in the background, collecting environmental data as the robot moves. This data collection process is synchronized with the robot's motion, ensuring that each measurement corresponds to a specific location in the environment. Although the LiDAR data does not directly influence the robot's motion during exploration, it plays a crucial role in subsequent mapping and analysis.

4.4 Map Processing Strategy

The map processing stage is a critical component of the proposed system, as it transforms the raw sensor data collected during exploration into a usable representation of the environment.

While the LiDAR sensor provides a large amount of distance measurements, these measurements in their original form are not directly suitable for visualization, analysis, or path planning. Therefore, a structured processing strategy is required to convert, organize, and refine the data.

During the exploration phase, the LiDAR continuously generates distance measurements corresponding to different directions around the robot. These measurements are collected over time as the robot moves along the boundary. As a result, the raw dataset consists of multiple scans that together represent the environment. However, this data may contain inconsistencies such as overlapping points, irregular spacing, and measurement noise.

To address these issues, the map processing strategy is designed to perform a sequence of transformations and refinements. The first step is to convert the raw LiDAR measurements into a spatial representation. The second step involves organizing the data into a consistent structure that can be visualized and analyzed. Finally, the processed data is prepared for use in coverage path planning.

The processing is performed externally using MATLAB, which provides powerful tools for numerical computation and data visualization. By separating data collection and processing, the system ensures that real-time navigation is not affected by computationally intensive tasks.

4.4.1 Coordinate-Based Mapping

The coordinate-based mapping method forms the core of the environment representation process in this study. It is responsible for transforming raw LiDAR measurements into a structured spatial representation that can be used for visualization, analysis, and path planning. This step acts as a bridge between sensor data acquisition and higher-level processing tasks.

The LiDAR sensor generates measurements in polar form, where each data point is defined by a distance and a corresponding angle relative to the sensor's orientation. While this representation is suitable for sensing, it does not directly provide spatial information in a format that can be easily interpreted or visualized. Therefore, a coordinate transformation is required to convert these measurements into Cartesian coordinates.

Each LiDAR measurement is transformed into an (x, y) coordinate relative to the sensor using trigonometric relationships. This conversion allows the position of each detected point to be expressed in a two-dimensional plane. As a result, each scan produces a set of points that represent the geometry of the environment from the robot's current position.

However, since the robot is continuously moving during exploration, each scan corresponds to a different position and orientation of the robot. To ensure consistency, the local coordinates obtained from the LiDAR must be transformed into a common reference frame. This transformation accounts for the robot's position and orientation at the time of each measurement, allowing all data points to be aligned within a global coordinate system.

The process of transforming local coordinates into a global frame is essential for building a coherent map. Without this step, the data points from different scans would remain isolated and would not accurately represent the structure of the environment. By applying consistent

transformations, the system accumulates data over time, gradually constructing a complete spatial representation.

As the robot continues its exploration, the number of data points increases, forming a dense point cloud. This point cloud outlines the boundaries of the environment, the shapes of obstacles, and the free space available for navigation. The density of the point cloud depends on factors such as the LiDAR resolution, scanning frequency, and robot speed. A higher resolution and slower movement generally result in a more detailed representation.

Despite its advantages, the coordinate-based mapping approach also introduces certain challenges. The raw point cloud may contain noise due to sensor inaccuracies, reflections, or environmental factors. Additionally, overlapping measurements may occur when the robot revisits the same area, leading to redundant data points. Variations in robot motion can also cause slight misalignment between scans.

To address these issues, the coordinate data is further processed in MATLAB. Filtering techniques are applied to remove noise and outliers, while data organization methods are used to improve the clarity of boundaries. In some cases, point clustering or smoothing may be applied to enhance the structure of the map.

Another important consideration is the representation of curved surfaces and complex geometries. The coordinate-based method is capable of capturing such features, as the LiDAR provides sufficient resolution to detect gradual changes in shape. This is particularly useful for representing curved boundaries, such as circular obstacles, which are commonly found in real-world environments.

The coordinate-based mapping approach is chosen for its balance between simplicity and effectiveness. Unlike grid-based or probabilistic mapping methods, it does not require extensive memory or computational resources. At the same time, it provides enough detail to support visualization and path planning tasks.

4.4.2 MATLAB-Based Map Processing

After converting LiDAR measurements into a coordinate-based representation, further processing is required to transform the raw point cloud into a structured and usable map. Although the coordinate-based mapping step provides a spatial representation of the environment, the resulting data often contains noise, redundancy, and irregularities that can affect subsequent analysis and path planning. Therefore, MATLAB is used as an external processing tool to refine and enhance the map.

The use of MATLAB in this stage is motivated by its powerful capabilities in numerical computation, data visualization, and signal processing. It allows efficient handling of large datasets and provides a wide range of built-in functions for filtering, transformation, and analysis. By transferring the LiDAR data to MATLAB, the system can perform complex processing tasks without burdening the real-time control system.

The first step in MATLAB-based processing is data import and visualization. The coordinate data obtained from the LiDAR is loaded into MATLAB and plotted in a two-dimensional plane. This initial visualization provides a direct view of the environment, allowing verification of data integrity and identification of potential issues such as missing regions or excessive noise.

Following visualization, noise reduction is performed to improve the quality of the map. Noise in the point cloud may arise from sensor inaccuracies, reflections, or rapid robot motion. These unwanted data points can obscure the true structure of the environment. Filtering techniques are applied to remove outliers and reduce variability in the data. This step enhances the clarity of boundaries and improves the reliability of the map.

Another important aspect of processing is the handling of redundant data. As the robot moves and revisits certain areas, multiple measurements may be recorded for the same region. While redundancy can improve accuracy, excessive overlapping points can increase computational load and reduce clarity. Therefore, data reduction techniques are applied to maintain an appropriate balance between detail and efficiency.

The organization of point data is also addressed during this stage. The raw point cloud may be unevenly distributed due to variations in robot speed and sensor scanning patterns. MATLAB is used to reorganize the data into a more uniform structure, which improves the consistency of the map and facilitates further processing.

In addition, boundary extraction can be performed to identify the outer limits of the environment. This involves detecting clusters of points that correspond to walls, curbs, and obstacles. By isolating these features, the system can generate a clearer representation of navigable space and constraints.

Another key function of MATLAB processing is the preparation of data for coverage path planning. The refined map is analyzed to determine free space and obstacle regions. This information is used to design a path that ensures complete coverage of the environment. The processed map must therefore be both accurate and structured to support this task.

The use of MATLAB also allows for easy visualization of intermediate and final results. Different stages of processing can be displayed and compared, providing insight into how the data evolves from raw measurements to a refined map. This capability is valuable for debugging, validation, and performance evaluation.

4.5 Cleaning Method Selection

The selection of an appropriate cleaning method is an essential aspect of the proposed system, as it determines how effectively the robot can cover the entire environment after the exploration and mapping phases. The cleaning method must ensure maximum area coverage while minimizing redundant movement and avoiding missed regions. Additionally, it must be compatible with the map representation generated from LiDAR data and the motion capabilities of the differential-drive robot.

In this study, the cleaning strategy is designed based on a combination of boundary-following and structured coverage methods. This hybrid approach is selected because the environment consists of both boundary regions and open interior areas, each requiring a different handling strategy. A single method alone is not sufficient to achieve complete and efficient coverage.

The first component of the cleaning strategy focuses on boundary regions. Since the robot already performs curb-following during the exploration phase, this behavior can be reused to ensure that all edges of the environment are properly covered. Boundary-following is particularly important because edges and corners are areas that are often missed by simple coverage patterns. By explicitly following the boundary, the robot ensures that these critical regions are included in the cleaning process.

The second component of the strategy addresses the interior of the environment. For this purpose, a structured coverage pattern is used, typically in the form of a back-and-forth or zigzag trajectory. This method is widely used in cleaning robots because it provides systematic and predictable coverage. The robot moves along parallel paths, covering the area in successive passes until the entire region is traversed.

The spacing between adjacent paths is a key parameter in this method. It must be selected based on the effective cleaning width of the robot to ensure that there is sufficient overlap between passes. If the spacing is too large, gaps may remain between paths, resulting in uncleaned regions. If the spacing is too small, the robot may perform redundant movements, reducing efficiency. Therefore, an appropriate balance is chosen to optimize coverage performance.

Another important consideration is the transition between boundary-following and interior coverage. After completing the boundary traversal, the robot must reposition itself to begin the structured coverage pattern. This transition is designed to be smooth and efficient, ensuring that the robot can seamlessly switch between different modes of operation.

The cleaning method also accounts for the presence of obstacles within the environment. During coverage, the robot must adjust its path to avoid obstacles while maintaining overall coverage. This may involve temporarily deviating from the planned path and then returning to continue the coverage pattern. The flexibility of the zigzag method allows such adjustments without significantly affecting overall performance.

The selection of this hybrid cleaning strategy is based on its simplicity, reliability, and effectiveness. It does not require complex optimization algorithms or extensive computational resources, making it suitable for implementation in systems with limited processing capabilities. At the same time, it provides a high level of coverage and adaptability to different environment geometries.

5 Implementation

This section provides an overview of how the proposed autonomous cleaning robot system is implemented in practice. The implementation translates the previously described design and methodological concepts into a functional system that operates in both simulation and real-world environments.

The implementation is structured in a way that separates the system into interconnected components, including navigation, sensing, control, and data processing. Each component is developed individually and then integrated to ensure coordinated operation. This approach allows for easier debugging, validation, and scalability of the system.

At the core of the implementation is the real-time control system, which governs the motion of the robot. This control system continuously receives input from ultrasonic sensors and generates motor commands to guide the robot along the curb and avoid obstacles. The control logic is designed to operate in a continuous loop, ensuring that the robot responds immediately to changes in its environment.

Simultaneously, the LiDAR sensor operates as a data acquisition module. As the robot moves, the LiDAR continuously scans the surroundings and collects distance measurements. These measurements are converted into spatial coordinates and stored for further processing. Unlike the ultrasonic sensors, which are used directly for navigation, the LiDAR data is primarily used for mapping and analysis.

The implementation also includes a communication mechanism that enables the transfer of LiDAR data from the robot system to an external processing environment. This is achieved through signal-based communication in simulation and serial communication in the real-world setup. The transmitted data is then processed in MATLAB to generate a map of the environment and design a coverage path.

Another important aspect of the implementation is the synchronization between different system components. The robot must maintain stable motion while continuously collecting data. This requires careful coordination between the control loop and the data acquisition process. The system is designed such that real-time control is not interrupted by data processing tasks, ensuring consistent and reliable operation.

The implementation is carried out in two main stages. The first stage involves simulation in CoppeliaSim, where the robot model, sensors, and environment are configured. This stage allows testing and validation of the system under controlled conditions. The second stage involves applying the same principles to a physical robot, verifying that the system performs similarly in real-world conditions.

5.1 Robot Control and Sensor Implementation

The robot control system is implemented using a Lua script attached to the Pioneer P3DX model in the CoppeliaSim environment. This script directly governs the robot's motion by processing ultrasonic sensor data and converting it into motor velocity commands in real time. The implementation follows a reactive control architecture, where all decisions are based on

current sensor readings without reliance on precomputed trajectories.

At the initialization stage, the script obtains references to the robot body and its actuators. The left and right motors are accessed as joint objects, enabling direct control over wheel velocities. Three ultrasonic sensors are also initialized using indexed object references. These sensors are positioned at the front, front-right, and right side of the robot, each serving a specific role in navigation.

The control behavior is defined using a set of experimentally tuned parameters. The base forward velocity is set to 2.0, which determines the nominal speed of the robot. The desired distance from the curb is defined as 0.35 meters, representing the target offset that the robot maintains during boundary tracking. A proportional gain of 2.0 is used to control the responsiveness of the system, and an obstacle detection threshold of 0.40 meters is defined for collision avoidance.

A helper function is implemented to simplify sensor data handling. This function reads the proximity sensor output and returns a valid distance when an object is detected. If no object is detected, a negative value is returned. This approach allows the control logic to clearly distinguish between valid and invalid measurements.

The main control logic is executed within the actuation function, which runs continuously during simulation. The control structure consists of three main layers: exploration completion detection, obstacle avoidance, and curb-following.

The exploration completion mechanism is based on the robot's displacement from its starting position. The initial position is stored at the beginning of the simulation. The robot must first move more than 1.0 meter away from this point to ensure that exploration has begun. Once this condition is satisfied, the system monitors whether the robot returns within 0.5 meters of the starting position. When this occurs, the robot is stopped by setting both motor velocities to zero, indicating that a full loop has been completed.

Obstacle avoidance is implemented as the highest-priority behavior. The front and front-right ultrasonic sensors are continuously monitored. If either sensor detects an object within the threshold distance of 0.40 meters, the robot immediately executes a turning maneuver. This is achieved by assigning a negative velocity to the left motor and a higher positive velocity to the right motor, causing the robot to rotate away from the obstacle. This condition overrides all other control actions to ensure safe operation.

If no obstacle is detected, the robot proceeds with curb-following behavior. When the right-side sensor fails to detect a boundary, indicated by a negative measurement, the robot assumes that it has drifted away from the curb. In response, it adjusts its motion by reducing the speed of the right motor while maintaining forward motion on the left motor, effectively steering back toward the boundary.

When valid distance data is available from the right-side sensor, the robot applies proportional control to maintain the desired distance from the curb. The error is calculated as

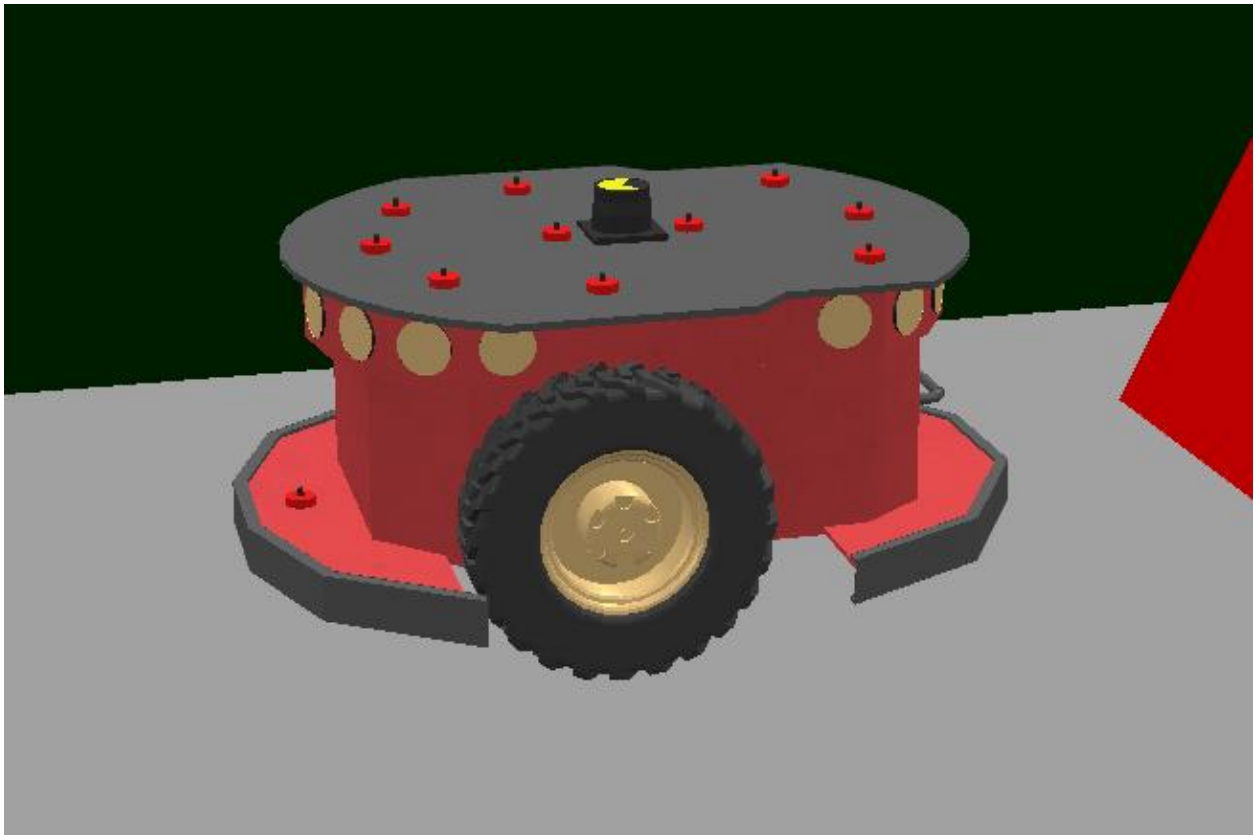
the difference between the desired distance and the measured distance. This error is multiplied by the proportional gain to produce a control signal that determines the turning intensity.

The control signal is then applied to the motor velocities. Specifically, it is subtracted from the left motor speed and added to the right motor speed. This creates a differential motion that corrects the robot's trajectory. If the robot is too close to the curb, it steers away; if it is too far, it steers toward the curb. This continuous adjustment results in smooth and stable boundary tracking.

The implemented control system achieves a balance between simplicity and functionality. The use of proportional control allows real-time operation with minimal computational overhead, while the inclusion of obstacle avoidance and completion detection ensures robustness. The behavior observed in simulation confirms that the robot can successfully follow the curb, avoid obstacles, and complete a full exploration cycle.

5.1.1 Robot Model Configuration

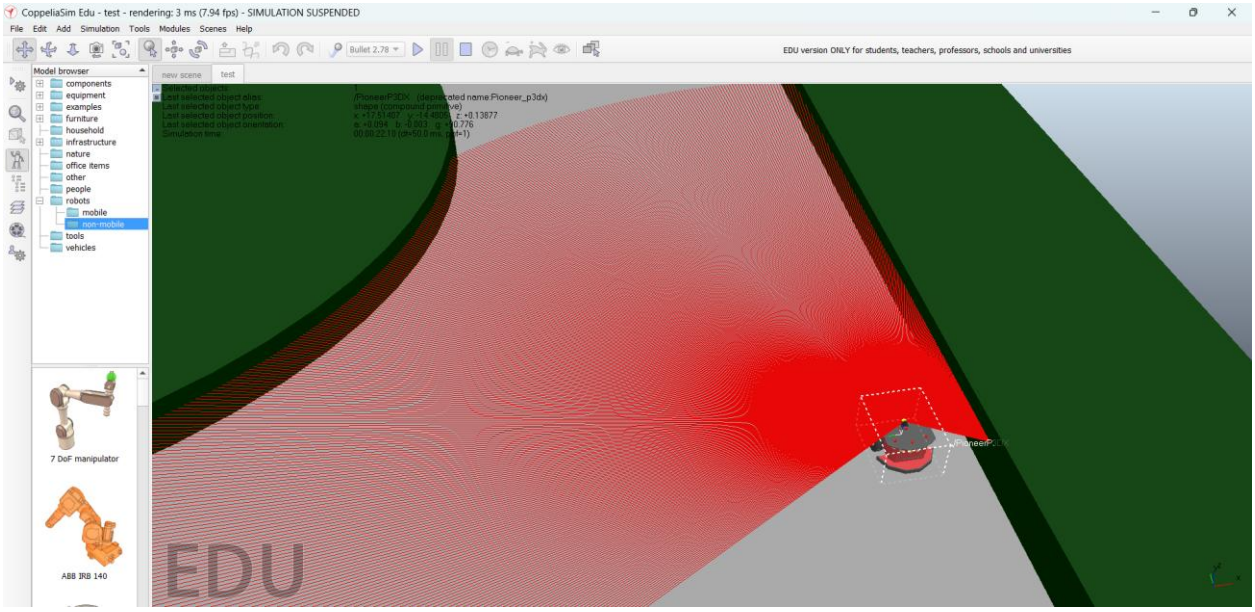
The Pioneer P3DX mobile robot model was selected as the primary platform for system implementation in the CoppeliaSim environment. This model was chosen due to its realistic physical characteristics, compatibility with differential-drive control, and availability within the built-in model library. The default geometric and dynamic parameters of the model were retained, as they closely approximate a real-world mobile robot used in indoor and semi-structured environments.



The robot is based on a differential-drive locomotion mechanism, which consists of two independently actuated drive wheels and a passive caster wheel for balance. This configuration provides a high degree of maneuverability while maintaining mechanical simplicity. The absence of complex steering mechanisms reduces control complexity and allows direct mapping between control inputs and robot motion.

From an implementation perspective, the left and right wheels are represented as revolute joints within CoppeliaSim. These joints are accessed programmatically using object handles obtained during the initialization phase of the script. The motion of the robot is controlled by assigning target angular velocities to these joints using the simulation API. This approach allows precise and continuous control over the robot's movement.

The kinematic behavior of the robot is governed by the relative velocities of the two drive wheels. When both wheels rotate at equal velocities, the robot moves forward in a straight line. When there is a difference between the velocities, the robot follows a curved trajectory, where the radius of curvature depends on the magnitude of the velocity difference. When the wheels rotate in opposite directions, the robot performs an in-place rotation about its central axis. This capability is particularly important for obstacle avoidance and reorientation in confined spaces.



The robot is initialized at a predefined position within the simulation environment. During the initialization phase, the position of the robot is retrieved using simulation functions and stored as a reference point. This initial position is later used to determine whether the robot has completed a full exploration cycle. By continuously calculating the Euclidean distance between the current position and the starting position, the system can detect loop closure and terminate exploration appropriately.

The dynamic behavior of the robot is also an important aspect of the implementation. The model is configured with physical properties such as mass, inertia, and friction coefficients that influence its interaction with the environment. Collision detection is enabled to ensure that the robot cannot pass through obstacles or boundaries. This is achieved by setting the environment and robot components as respondable objects within the simulation. As a result, realistic physical interactions are maintained during operation.

Another critical consideration is the coordinate system used in the simulation. The robot operates within a global reference frame defined by the simulation environment, while its motion and sensor data are often expressed in a local coordinate frame attached to the robot body. The transformation between these coordinate systems is handled internally within the simulation and explicitly in the sensor processing scripts. This distinction is important for accurate data interpretation and mapping.

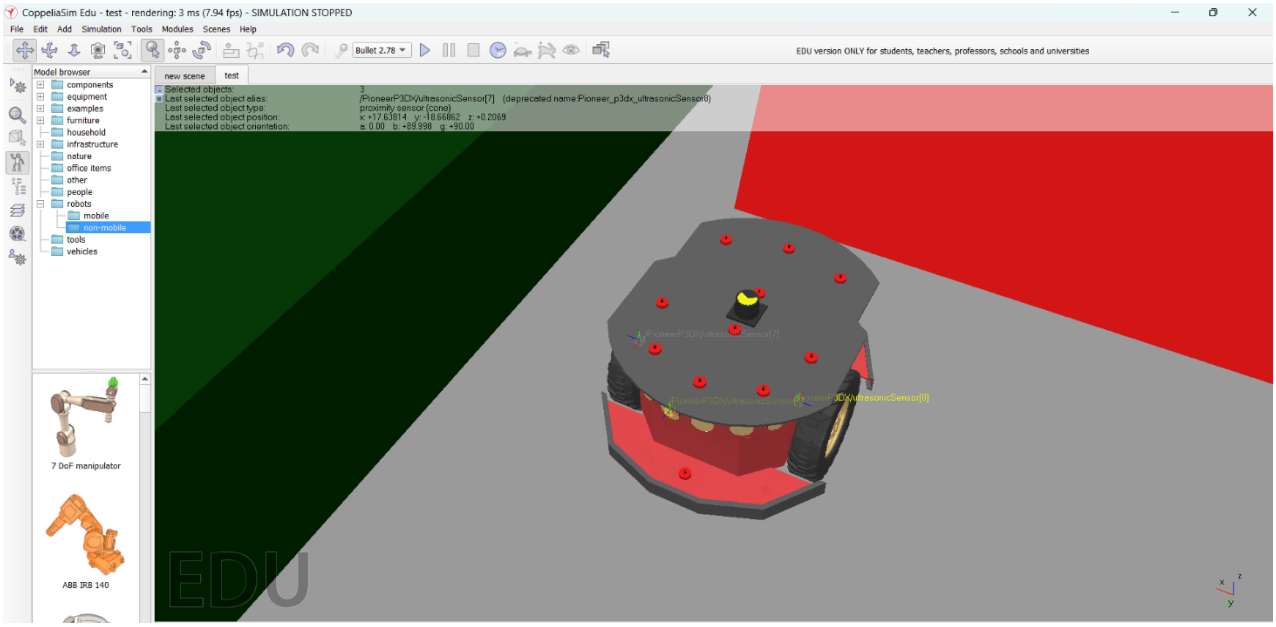
The robot model also serves as the integration platform for all sensing components. The ultrasonic sensors and LiDAR system are mounted on the robot body in positions that maximize coverage and minimize interference. The placement of sensors is carefully designed to ensure that their fields of view are not obstructed by the robot structure. The right-side ultrasonic sensor is positioned to maintain consistent distance measurements from the curb, while the front sensors provide early detection of obstacles. The LiDAR sensor is mounted at an elevated position to achieve a wide scanning range.

In addition to structural configuration, the robot model supports modular expansion. The implementation allows additional sensors or control modules to be integrated without requiring significant modification to the existing system. This flexibility is important for future development, such as incorporating advanced localization or mapping algorithms.

The simulation time step and execution cycle also play a role in the robot’s behavior. The control script operates within the simulation loop, where sensor readings and motor commands are updated at each time step. The selected time step ensures a balance between computational efficiency and control accuracy. A sufficiently high update rate allows smooth motion and responsive control, which is essential for stable navigation.

5.1.2 Ultrasonic Sensor Implementation

The ultrasonic sensing system is implemented as the primary real-time perception component for navigation and obstacle avoidance. In the CoppeliaSim environment, ultrasonic sensors are simulated using proximity sensors, which provide distance measurements to nearby objects within a defined detection range and field of view.



A total of three ultrasonic sensors are integrated into the robot model. These sensors are positioned strategically to support different navigation functions. The first sensor is mounted at the front of the robot, aligned with the forward direction. This sensor is responsible for detecting obstacles directly in the robot's path and plays a critical role in collision prevention. The second sensor is positioned at a front-right angle, allowing early detection of obstacles that are not directly in front but may intersect the robot's trajectory. The third sensor is mounted on the right side of the robot and is used for maintaining a consistent distance from the curb during boundary-following behavior.

The sensors are accessed in the control script using object handles obtained during the initialization phase. Each sensor is referenced using an indexed access method, which allows multiple sensors of the same type to be managed efficiently. This indexing approach simplifies the implementation and ensures that sensor data can be read in a structured manner.

During each simulation step, the control script reads data from the ultrasonic sensors using the simulation API function for proximity sensing. The raw sensor output includes both a detection state and a measured distance. To simplify the control logic, a helper function is implemented to process this data. If an object is detected, the function returns the measured distance. If no object is detected, a negative value is returned. This allows the control algorithm to easily distinguish between valid and invalid measurements without additional conditional checks.

The detection range and field of view of the sensors are configured to match realistic ultrasonic sensor behavior. The range is selected to ensure that the robot has sufficient time to react to obstacles, while the field of view determines the angular coverage of each sensor. The combination of three sensors provides partial coverage of the robot's surroundings, which is sufficient for reactive navigation.

Each sensor plays a specific role within the control system. The front sensor provides immediate feedback for obstacle avoidance. When an object is detected within a predefined threshold distance, the robot initiates a turning maneuver to avoid collision. The front-right sensor enhances this capability by detecting obstacles earlier and providing additional information about their position relative to the robot.

The right-side sensor is used for curb-following behavior. It continuously measures the lateral distance between the robot and the boundary. This measurement is used as input to the proportional control algorithm, which adjusts the robot's trajectory to maintain a desired distance from the curb. The accuracy and consistency of this sensor are critical for stable navigation.

The update frequency of the sensors is aligned with the simulation time step, ensuring that new measurements are available at each control cycle. This allows the robot to respond quickly to changes in the environment and maintain smooth motion.

Despite their advantages, ultrasonic sensors have certain limitations. Their measurements can be affected by surface properties, angle of incidence, and environmental noise. For example, objects with irregular shapes or soft surfaces may produce weaker reflections, leading to less accurate distance readings. Additionally, the sensors provide limited angular resolution, which restricts their ability to capture detailed spatial information.

To address these limitations, the ultrasonic sensors are used primarily for local, real-time decision-making, while the LiDAR sensor is used for global data collection and mapping. This division of roles ensures that each sensor type is used according to its strengths.

5.1.3 Control Algorithm Implementation

The control algorithm of the robot is implemented using a Lua script within the CoppeliaSim environment. This algorithm is responsible for converting sensor measurements into motor commands that govern the robot's motion. The implementation follows a reactive control approach, where decisions are made continuously based on real-time sensor data.

The control logic is executed inside the actuation callback function, which runs at every simulation step. During each cycle, the system reads sensor inputs, evaluates control conditions, and updates the velocities of the left and right motors accordingly. This loop-based execution ensures continuous and responsive behavior.

At the beginning of the script, several control parameters are defined. The base speed of the robot is set to 2.0, which determines the nominal forward velocity. The desired distance from the curb is defined as 0.35 meters, and this value serves as the reference for boundary tracking. A proportional gain of 2.0 is used to control the responsiveness of the system, while an obstacle detection threshold of 0.40 meters is defined for collision avoidance.

The control algorithm is structured into multiple layers of behavior, each with a specific priority. The highest priority is given to obstacle avoidance. The front and front-right ultrasonic sensors are continuously monitored for obstacles. If either sensor detects an object within the threshold distance, the robot immediately executes a turning maneuver. This is achieved by

assigning a negative velocity to one motor and a higher positive velocity to the other, causing the robot to rotate away from the obstacle. This behavior overrides all other control actions to ensure safe operation.

If no obstacle is detected, the algorithm proceeds to boundary-following behavior. The distance measured by the right-side ultrasonic sensor is compared with the desired distance. The difference between these values is calculated as an error term. This error is then multiplied by the proportional gain to produce a control signal.

The control signal is used to adjust the motor velocities. Specifically, the signal is subtracted from the left motor speed and added to the right motor speed. This creates a differential motion that allows the robot to steer toward or away from the curb. If the robot is too close to the boundary, it turns away; if it is too far, it turns toward the boundary. This continuous adjustment enables smooth and stable curb-following behavior.

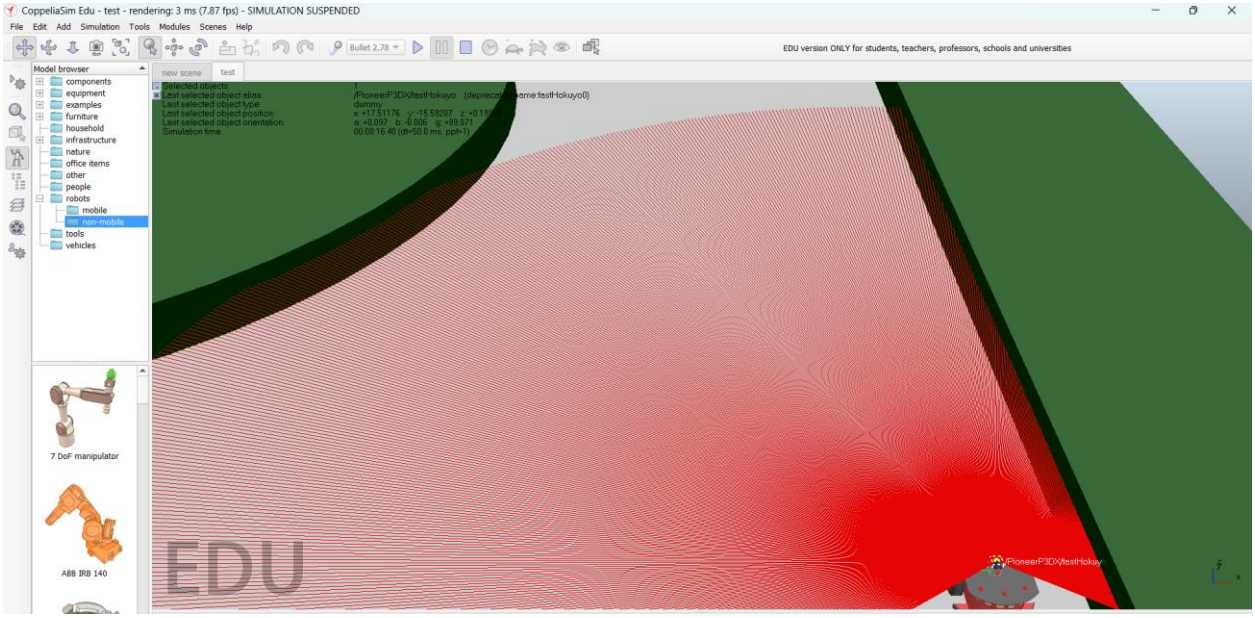
An additional condition is implemented to handle situations where the boundary is not detected by the right-side sensor. In this case, the sensor returns a negative value, indicating that no object is within range. The algorithm interprets this as the robot drifting away from the curb and responds by adjusting the motor speeds to steer the robot back toward the boundary.

The control algorithm also includes a mechanism to determine when the exploration phase is complete. The initial position of the robot is stored at the beginning of the simulation. During operation, the robot continuously calculates its distance from this starting point. The system first ensures that the robot has moved beyond a minimum distance to avoid premature termination. Once this condition is satisfied, the algorithm monitors whether the robot returns close to its starting position. When both conditions are met, the robot stops, indicating that a full loop around the environment has been completed.

The implementation of the control algorithm emphasizes simplicity and efficiency. The use of proportional control allows real-time execution with minimal computational overhead, while still providing sufficient accuracy for navigation. The hierarchical structure of behaviors ensures that critical conditions, such as obstacle avoidance, are handled immediately.

5.2 LiDAR Implementation and Data Acquisition

The LiDAR sensing system is implemented in the CoppeliaSim environment using a fast Hokuyo URG-type configuration. Instead of using a single physical LiDAR model, the system utilizes two vision sensors combined with a custom Lua script to simulate scanning behavior and generate spatial data. This approach provides flexibility in controlling scan parameters and enables direct access to processed coordinate data.



During initialization, key parameters of the LiDAR system are defined. The maximum scan distance is set to 5 meters, which limits the sensing range and ensures that only relevant environmental features are captured. The system is configured to generate coordinate-based data by disabling range-only output. This allows each detected point to be represented directly in Cartesian coordinates rather than as distance values.

The LiDAR sensors are configured to detect objects within a specified collection in the simulation. This ensures that only relevant entities, such as walls and obstacles, are included in the sensing process, while the robot's own structure is excluded. The far clipping parameter of each vision sensor is adjusted to match the defined maximum scanning distance.

The sensing process is executed within the sensing callback function, which runs at each simulation step. During each cycle, the script reads data from both vision sensors using the simulation API. The returned data includes directional vectors and distance values for each detected point within the sensor's field of view.

To convert this data into usable spatial information, a coordinate transformation is performed. Each detected point is initially expressed in the local coordinate system of the sensor. A transformation matrix is computed to map these points into the robot's reference frame. This transformation ensures that all data points are aligned consistently, regardless of the sensor's orientation.

The transformation process involves obtaining the transformation matrix of the sensor and the robot, computing the relative transformation between them, and applying this transformation to each detected point. As a result, the coordinates of all points are expressed relative to the robot's position.

For each valid measurement, the x, y, and z coordinates are extracted and stored in a data structure. These values are appended sequentially to form a data array representing the point cloud generated during the current scan cycle. Points that exceed the maximum scanning distance can be optionally discarded to improve data quality and reduce noise.

In addition to data acquisition, the implementation includes a visualization mechanism. A drawing object is created within the simulation, and each detected point is rendered as a line segment. This provides a real-time visual representation of the LiDAR scan, allowing verification of sensor performance and alignment.

The collected data represents a dense set of spatial points describing the surrounding environment. As the robot moves, multiple scans are accumulated, resulting in a comprehensive point cloud that captures boundaries, obstacles, and free space.

Although the script primarily focuses on data generation, the resulting coordinate data can be further processed or transmitted for external analysis. In extended implementations, the data can be packed into a signal and exported for use in MATLAB, where it can be used to generate maps and perform further processing.

5.3 LiDAR Data Extraction and MATLAB Processing

This section describes how the LiDAR data generated in CoppeliaSim is extracted, formatted, and processed in MATLAB to produce a usable map of the environment. Unlike the previous section, which focused on sensor implementation, this section focuses on the actual data pipeline from simulation to map generation.

During simulation, the LiDAR script generates spatial data in the form of Cartesian coordinates. Each sensing cycle produces a set of points expressed as (x, y, z) values relative to the robot's coordinate frame. These values are stored sequentially in a data array, forming a point cloud representation of the environment.

To make this data usable outside the simulation, it is transmitted using the CoppeliaSim signal mechanism. The coordinate array is first converted into a packed string format using a float packing function. This packed data is then assigned to a string signal, typically named lidarData. This allows other scripts or external interfaces to access the LiDAR data in real time.

On the receiving side, the data is retrieved using a signal reading function. The packed string is then unpacked back into a numerical array containing the coordinate values. The resulting data structure consists of sequential x, y, and z values, which can be grouped into individual points.

Since the mapping process is focused on two-dimensional representation, only the x and y coordinates are used. The z coordinate is ignored, as the robot operates on a planar surface. The extracted (x, y) pairs represent the positions of detected objects relative to the robot.

As the robot moves through the environment, LiDAR data is continuously collected over multiple time steps. These measurements are accumulated to form a complete dataset representing the entire environment. This accumulation process is essential for generating a global map, as a single scan only captures a limited portion of the surroundings.

The collected data is then exported and processed in MATLAB. The coordinate values are loaded into MATLAB and visualized using scatter plots. Each point is plotted based on its x and y values, resulting in a point cloud map that represents the structure of the environment.

However, the raw data often contains noise and redundant points. Noise may be caused by sensor limitations or reflections, while redundancy occurs when the robot revisits the same area multiple times. To address these issues, filtering techniques are applied in MATLAB. Outliers are removed, and closely spaced duplicate points are reduced to improve clarity.

Another important step is coordinate alignment. Since the LiDAR data is initially expressed relative to the robot's frame, it may need to be transformed into a global coordinate system using the robot's position and orientation at each time step. This ensures that all points are correctly aligned and form a coherent map.

After processing, the resulting map provides a clear representation of the environment, including boundaries, obstacles, and free space. This map serves as the foundation for further tasks such as coverage planning and performance evaluation.

The use of MATLAB for processing allows efficient handling of large datasets and provides powerful visualization tools. It also enables additional analysis, such as measuring distances, identifying obstacles, and generating coverage paths.

5.4 Mission Execution and System Evaluation

This section describes how the implemented system is executed in the simulation environment and how its performance is evaluated. After completing the control implementation and LiDAR data processing stages, the robot operates as a fully integrated system that performs exploration, data collection, and trajectory execution.

The execution process begins by initializing the robot at a predefined starting position within the simulation environment. Once the simulation starts, the control script becomes active, and the robot begins moving forward using the defined base speed. At this stage, the ultrasonic sensors continuously provide real-time feedback, allowing the robot to follow the curb and avoid obstacles.

During motion, the LiDAR system operates simultaneously in the background. It continuously scans the environment and generates coordinate data representing surrounding objects. This data is collected at each simulation step and forms a growing dataset as the robot moves through the environment.

The robot follows a complete loop around the environment using the implemented curb-following algorithm. The exploration completion condition is monitored throughout execution. The robot must first move a sufficient distance away from the starting position and then return

close to it. Once both conditions are satisfied, the system identifies that the exploration phase is complete and stops the robot.

The collected LiDAR data is then processed externally in MATLAB to generate a map. This map is used to analyze the structure of the environment and verify that the robot has successfully captured relevant features such as boundaries and obstacles.

To evaluate system performance, several aspects are considered. The first is navigation stability. During execution, the robot maintains a relatively consistent distance from the curb, indicating that the proportional control algorithm is functioning correctly. Minor deviations may occur in curved sections, but the robot is able to correct its trajectory without oscillation.

The second aspect is obstacle avoidance performance. The robot successfully detects obstacles using the front and front-right sensors and adjusts its motion to avoid collisions. The transition between obstacle avoidance and normal operation is smooth, demonstrating proper prioritization within the control logic.

The third aspect is data acquisition quality. The LiDAR system generates a dense set of points that accurately represent the environment. The resulting point cloud clearly outlines the boundaries and obstacles, confirming that the sensor implementation is effective.

Another important factor is execution reliability. The robot is able to complete a full loop without becoming stuck or losing the boundary. This indicates that the control system is robust and capable of handling different environmental conditions.

Although the system performs effectively, some limitations are observed. Small inaccuracies in sensor readings can lead to minor deviations in trajectory. Additionally, the reliance on proportional control introduces a small steady-state error in maintaining the desired distance from the curb.

5.5 Cleaning Path Generation

This section describes the implementation of the cleaning path generation process based on the map obtained from LiDAR data and processed in MATLAB. The purpose of this stage is to convert the processed map into a sequence of executable waypoints that enable the robot to cover the entire environment efficiently.

The cleaning path is generated using a structured coverage strategy that combines boundary coverage and interior area coverage. This approach ensures that both edges and open spaces within the environment are effectively traversed.

The first step in the path generation process is identifying the navigable area from the processed map. The map consists of coordinate-based data representing obstacles and free space. The free-space region is extracted and used as the valid area for path planning. Any regions that are too close to obstacles are excluded to maintain a safe distance during robot operation.

Once the valid area is determined, boundary paths are generated along the edges of the environment. These paths follow the contours of walls and obstacles, ensuring that regions near

boundaries are not left uncleaned. Since the robot cannot move directly along the boundary line, the paths are offset inward by a fixed distance corresponding to the robot's size and safety margin. This offset ensures that the robot can follow the path without collision.

After generating boundary paths, the interior cleaning paths are created using a structured zigzag pattern. This pattern consists of parallel lines that cover the free space systematically. The spacing between adjacent lines is determined based on the effective cleaning width of the robot. Proper spacing ensures that the entire area is covered without leaving gaps or causing excessive overlap.

The zigzag paths are generated by dividing the free-space region into parallel strips. Each strip is converted into a sequence of waypoints, which the robot can follow sequentially. The direction of motion alternates between consecutive strips, allowing the robot to move continuously without unnecessary turning.

To ensure efficient execution, the generated paths are ordered and connected. The boundary paths are executed first, followed by the interior zigzag paths. Transition paths are generated between consecutive segments to allow smooth movement from one path to another. These transition paths are designed to avoid obstacles and remain within the navigable area.

The final output of this process is a complete set of waypoints representing the cleaning mission. Each waypoint contains position coordinates that define the path to be followed by the robot. These waypoints are exported as a structured data file, which can be loaded into the simulation environment for execution.

During execution, the robot follows these waypoints using its control system. The path-following behavior ensures that the robot moves along the planned trajectory while maintaining stability and avoiding obstacles.

The implemented cleaning path generation method provides a balance between simplicity and effectiveness. It does not rely on complex optimization algorithms but still achieves high coverage of the environment. The combination of boundary following and structured interior coverage ensures that both edge regions and open areas are included in the cleaning process.

6 Results and Discussion

This chapter presents and analyzes the results obtained from the implementation of the autonomous cleaning robot system. The objective of this chapter is to evaluate how effectively the developed system performs in terms of navigation, sensing, mapping, and cleaning coverage.

The evaluation is based on experiments conducted in the CoppeliaSim environment, where the robot operates within a structured map containing straight boundaries, curved paths, and obstacles. During execution, the robot performs exploration using ultrasonic sensor-based control while simultaneously collecting environmental data using the LiDAR system. The collected data is then processed in MATLAB to generate a map and cleaning path.

The results presented in this chapter are divided into several key components. First, the navigation performance is analyzed, focusing on the robot's ability to follow the curb, maintain stability, and avoid obstacles. Second, the LiDAR data is evaluated to determine how accurately it represents the environment. Third, the processed map is examined to assess its clarity and correctness. Finally, the cleaning coverage performance is analyzed based on how effectively the robot traverses the environment.

In addition to presenting the results, this chapter also includes a discussion of system performance and limitations. The strengths of the system are identified, and areas for improvement are highlighted. This analysis provides insight into how the system can be enhanced in future work.

6.1 Navigation Performance Analysis

This section presents a detailed evaluation of the robot's navigation performance during the exploration phase. The analysis focuses on curb-following accuracy, obstacle avoidance behavior, trajectory stability, and overall motion consistency.

The robot navigation is primarily governed by a proportional control algorithm using feedback from the right-side ultrasonic sensor. The desired distance from the curb is set to 0.35 meters, and the system continuously adjusts motor velocities to maintain this reference.

In straight boundary segments, the robot demonstrates stable and consistent behavior. The measured distance from the curb remains close to the desired value, with only minor fluctuations. These fluctuations are typically within a small tolerance range and are caused by sensor resolution and control response delay. The robot maintains a smooth trajectory without noticeable oscillations, indicating that the selected proportional gain provides an appropriate balance between responsiveness and stability.

In curved sections of the environment, the robot's behavior becomes more dynamic. As the curvature of the boundary changes, the robot must continuously adjust its heading. The proportional controller responds to changes in measured distance, resulting in slight oscillatory corrections. In some cases, the robot temporarily deviates from the desired distance before converging back to the reference. This behavior is expected in proportional control systems, especially when handling continuous curvature.

The obstacle avoidance mechanism is evaluated based on the robot's response to objects detected by the front and front-right ultrasonic sensors. When an obstacle enters the detection range (0.40 meters), the robot immediately overrides its normal control behavior and performs a turning maneuver. This response is generally effective, preventing collisions and allowing the robot to continue navigation.

The transition between obstacle avoidance and curb-following is smooth in most cases. After avoiding an obstacle, the robot gradually returns to its boundary-following path without abrupt motion changes. However, minor delays are observed in some situations where obstacles are detected near the edge of the sensor range. These delays are attributed to the discrete update cycle of the control loop and the limited field of view of the sensors.

Trajectory stability is another important aspect of navigation performance. The robot's path is generally smooth, with gradual changes in direction. Sudden changes in velocity are minimized due to the continuous nature of the control algorithm. This contributes to stable motion and reduces the likelihood of mechanical instability.

The system also includes a mechanism to detect completion of exploration based on the robot's return to its starting position. This mechanism functions as expected, ensuring that the robot completes a full loop around the environment before stopping. The combination of distance threshold and return condition prevents premature termination of the exploration phase.

Despite the overall positive performance, certain limitations are observed. The proportional controller introduces a steady-state error, meaning that the robot does not always maintain the exact desired distance from the curb. Additionally, the response of the system is influenced by sensor noise and environmental geometry, which can cause small deviations in trajectory.

6.2 LiDAR Data and Mapping Results

This section presents a detailed analysis of the LiDAR data collected during the exploration phase and evaluates the quality of the generated map after processing in MATLAB. The goal is to assess how accurately the system captures and represents the environment.

During robot operation, the LiDAR system continuously scans the surroundings and generates a set of spatial points in the form of (x, y, z) coordinates. These points are produced at each simulation step and accumulated over time as the robot moves through the environment. The resulting dataset forms a point cloud that represents the geometry of the surroundings.

The raw LiDAR data shows a dense distribution of points around the robot's trajectory. Objects located closer to the robot appear as tightly clustered points, while objects at greater distances appear more sparsely distributed. This behavior is consistent with the characteristics of range-based sensing systems.

The boundaries of the environment are clearly visible in the raw point cloud. Straight walls appear as linear clusters of points, while curved boundaries form smooth arcs. The central circular obstacle is also well represented, appearing as a continuous curved shape. This indicates that the LiDAR implementation is capable of capturing both linear and nonlinear geometries effectively.

However, the raw data contains certain imperfections. Noise is present in the form of scattered points that do not correspond to actual surfaces. These may be caused by measurement uncertainty or simulation artifacts. In addition, overlapping points are observed in areas where the robot revisits the same location, resulting in redundancy within the dataset.

To improve the quality of the map, the raw data is processed in MATLAB. The first step involves extracting only the relevant (x, y) coordinates, as the mapping task is performed in a

two-dimensional plane. The data is then visualized using scatter plots, providing an initial representation of the environment.

Filtering techniques are applied to remove noise and reduce redundancy. Outliers are eliminated, and closely spaced duplicate points are reduced. This results in a cleaner and more structured point cloud. The boundaries become more clearly defined, and the overall shape of the environment is easier to interpret.

Another important aspect of the mapping process is coordinate consistency. Since the LiDAR data is initially expressed relative to the robot's frame, proper alignment is required to ensure that all points form a coherent global map. This is achieved by considering the robot's position during data collection and ensuring consistent transformation of coordinates.

The processed map provides a clear representation of the environment, including walls, obstacles, and free space. The geometric features observed in the simulation environment are accurately reflected in the map. Straight boundaries appear as well-defined lines, while curved structures maintain their shape with minimal distortion.

The effectiveness of the mapping process can also be evaluated based on completeness. The robot successfully captures data from all accessible regions of the environment, resulting in a nearly complete map. Minor gaps may exist in areas that are difficult to reach or where sensor coverage is limited, but these do not significantly affect the overall representation.

6.3 Cleaning Coverage Results

This section evaluates the cleaning coverage performance of the robot based on the generated path and the processed map. The objective is to determine how effectively the robot is able to traverse and cover the entire environment after completing the exploration and mapping phases.

The cleaning process begins after the map is generated in MATLAB and the coverage path is created using a structured approach. The path consists of two main components: boundary coverage and interior coverage. The robot first follows the boundary of the environment to ensure that edge regions are covered, and then proceeds to cover the interior area using a zigzag motion pattern.

During execution, the robot follows the generated waypoints sequentially. The movement between waypoints is controlled using the same differential-drive mechanism and sensor-based feedback system. The robot maintains stable motion while transitioning between different segments of the path.

The boundary coverage phase demonstrates high accuracy. The robot is able to follow the edges of the environment and navigate around obstacles, ensuring that areas close to walls and boundaries are included in the cleaning process. This is particularly important because boundary regions are often missed in simple coverage strategies.

The interior coverage phase is implemented using parallel trajectories arranged in a zigzag pattern. The spacing between these paths is determined based on the effective cleaning width of the robot. This spacing ensures that there is sufficient overlap between adjacent paths, minimizing the possibility of uncleaned areas.

The results show that the robot achieves a high level of coverage across the environment. Most of the accessible area is traversed, and the path design ensures systematic coverage without large gaps. The zigzag pattern proves effective in covering open spaces efficiently.

However, some limitations are observed in the coverage results. Small gaps may remain in regions near complex obstacles or sharp corners. These gaps are primarily caused by the physical constraints of the robot and the limitations of the path-following accuracy. In narrow passages, slight deviations in trajectory may also lead to partial overlap or missed regions.

Another factor affecting coverage performance is the accuracy of the generated map. Any noise or distortion in the map can influence the placement of coverage paths. Although MATLAB processing improves map quality, minor inaccuracies may still propagate into the path planning stage.

The efficiency of the coverage process is also considered. The structured path reduces redundant movement compared to random exploration methods. The robot follows a predictable trajectory, which minimizes unnecessary turns and improves overall efficiency.

Overall, the cleaning coverage results demonstrate that the implemented system is capable of achieving near-complete area coverage in a structured environment. The combination of boundary-following and zigzag coverage provides an effective balance between completeness and efficiency.

These results confirm that the proposed path generation and execution methods are suitable for autonomous cleaning applications, while also highlighting areas for potential refinement in future work.

6.4 System Performance Evaluation

This section provides an overall evaluation of the autonomous cleaning robot system by analyzing how well the integrated components perform together. The evaluation considers navigation accuracy, sensing reliability, mapping effectiveness, and overall system efficiency.

From a navigation perspective, the robot demonstrates stable and consistent motion throughout the exploration and cleaning phases. The proportional control algorithm enables the robot to maintain a reasonable distance from the curb while adapting to changes in the environment. Although minor deviations are observed in curved sections, the robot is able to correct its trajectory without instability. The obstacle avoidance mechanism also functions effectively, preventing collisions and allowing uninterrupted operation.

In terms of sensing performance, the combination of ultrasonic sensors and LiDAR provides a balanced perception system. The ultrasonic sensors offer reliable short-range measurements for real-time control, while the LiDAR system provides detailed spatial data for mapping. This division of sensing roles enhances the overall robustness of the system.

However, the limitations of ultrasonic sensors, such as limited resolution and sensitivity to surface properties, introduce small uncertainties in navigation.

The mapping performance is evaluated based on the quality of the generated point cloud and processed map. The LiDAR system successfully captures the structure of the environment, including boundaries and obstacles. After processing in MATLAB, the map becomes clearer and more structured, making it suitable for path generation. The coordinate-based mapping approach proves to be effective for representing the environment without requiring complex algorithms.

The cleaning coverage performance reflects the effectiveness of the path generation method. The combination of boundary-following and zigzag coverage allows the robot to traverse most of the environment systematically. The coverage is efficient, with minimal redundant movement, and the robot is able to complete the task within a reasonable time frame.

System integration is another important aspect of performance. The navigation and sensing components operate simultaneously without interfering with each other. The control system maintains real-time responsiveness, while the LiDAR continuously collects data in the background. This parallel operation demonstrates that the system architecture is well-structured and capable of handling multiple tasks.

Despite the overall positive performance, some limitations affect the system. The use of a proportional controller results in steady-state error, and sensor limitations introduce small inaccuracies. Additionally, the reliance on simulation means that certain real-world factors, such as noise and mechanical imperfections, are not fully represented.

6.5 Discussion and Limitations

This section provides a critical discussion of the overall system performance based on the results obtained from simulation and data processing. While the system demonstrates successful operation and achieves its primary objectives, several observations highlight both its strengths and limitations.

One of the key strengths of the system is its ability to achieve autonomous navigation using a relatively simple control strategy. The proportional control method, combined with ultrasonic sensor feedback, enables stable curb-following behavior without requiring complex computation. This makes the system efficient and suitable for real-time implementation on embedded platforms.

Another important strength is the integration of LiDAR-based sensing for environmental mapping. The implemented approach successfully generates coordinate-based data that accurately represents the structure of the environment. The use of MATLAB for processing further enhances map clarity, allowing effective path generation for cleaning tasks.

The modular architecture of the system also contributes to its robustness. Navigation and sensing operate independently, allowing real-time control to remain responsive while data collection continues in parallel. This separation simplifies system design and improves scalability for future extensions.

Despite these strengths, several limitations are identified. The most significant limitation is related to the control algorithm. The use of a proportional controller introduces steady-state error, meaning that the robot cannot always maintain the exact desired distance from the curb. Additionally, the controller may produce small oscillations, particularly in curved sections of the environment.

The ultrasonic sensors also present limitations. Their limited angular resolution and sensitivity to surface conditions can result in inaccurate or inconsistent measurements. This affects the precision of navigation, especially in environments with complex geometries or irregular surfaces.

The LiDAR implementation, while effective, is based on simulation using vision sensors. As a result, it does not fully replicate real-world sensor behavior, such as measurement noise, latency, and environmental interference. This may lead to differences between simulated and real-world performance.

The mapping process introduces additional challenges. The point cloud generated from LiDAR data requires post-processing to remove noise and redundancy. While MATLAB processing improves the quality of the map, some loss of detail or slight distortions may still occur. Furthermore, the absence of advanced mapping techniques such as SLAM limits the system’s ability to perform real-time localization and mapping.

Another limitation is related to the simulation environment itself. Although the simulation provides a controlled and efficient testing platform, it simplifies certain physical phenomena such as wheel slip, friction variation, and sensor noise. These factors can influence robot behavior in real-world scenarios and are not fully captured in the simulation.

6.5.1 System Limitations

Although the implemented system demonstrates effective performance in navigation, mapping, and coverage, several limitations are identified through experimental evaluation. These limitations are primarily related to control strategy, sensor characteristics, mapping approach, and simulation constraints.

One of the main limitations lies in the control algorithm. The system uses a proportional (P) controller for curb-following, which inherently introduces steady-state error. As a result, the robot does not always maintain the exact desired distance from the curb. In addition, the controller may produce small oscillations, particularly in regions with continuous curvature. This behavior is due to the lack of integral and derivative components, which could otherwise improve accuracy and damping.

Another limitation is associated with the ultrasonic sensors used for navigation. These sensors have a limited field of view and low angular resolution, which restricts their ability to accurately detect complex geometries. The measurements can also be affected by surface properties and angles, leading to inconsistent readings. In some cases, the sensors may fail to detect boundaries correctly, causing temporary deviations in the robot's trajectory.

The LiDAR implementation also has certain constraints. In this system, the LiDAR is simulated using vision sensors rather than a physical device. While this approach provides flexibility, it does not fully capture real-world characteristics such as measurement noise, latency, and environmental interference. Consequently, the accuracy observed in simulation may be higher than what can be achieved in practical applications.

The mapping process introduces additional limitations. The generated point cloud may contain noise and redundant data due to repeated scans of the same area. Although MATLAB processing reduces these issues, some inaccuracies may remain. Furthermore, the mapping approach is purely coordinate-based and does not include advanced techniques such as SLAM. This limits the system's ability to perform real-time localization and map correction.

Another limitation is related to the coverage path generation. The zigzag pattern, while effective in open areas, may not fully cover regions near complex obstacles or sharp corners. These areas can result in small gaps due to the robot's physical dimensions and path-following constraints.

Finally, the use of a simulation environment introduces inherent limitations. Factors such as wheel slip, uneven surfaces, and real sensor noise are simplified or not fully represented. As a result, the system's performance in simulation may differ from real-world behavior.

7 Conclusion and Future Work

7.1 Conclusion

This research presented the development of an autonomous cleaning robot system capable of performing curb-following navigation, environmental sensing, mapping, and area coverage. The objective of the study was to design a system that can explore an environment, understand its structure, and subsequently generate a path to cover the entire area efficiently.

The system was implemented using the Pioneer P3DX mobile robot in the CoppeliaSim environment. A reactive navigation approach was adopted, where ultrasonic sensors were used to provide real-time feedback for motion control. The robot successfully maintained a stable distance from the curb using a proportional control algorithm and demonstrated effective obstacle avoidance behavior through front-facing sensors.

A LiDAR-based sensing system was developed using vision sensors and custom scripting. This implementation enabled the robot to collect spatial data in the form of coordinate points. The collected data was processed in MATLAB to generate a map of the environment. The resulting map accurately represented boundaries and obstacles, providing a reliable basis for further processing.

Based on the generated map, a cleaning path was created using a combination of boundary-following and structured zigzag coverage. This approach ensured that both edge regions and interior areas were effectively covered. The robot was able to execute the generated path and achieve high coverage of the environment.

The results obtained from simulation demonstrate that the proposed system is capable of performing autonomous navigation, mapping, and cleaning tasks in a structured environment. The integration of control, sensing, and data processing components proved to be effective and reliable.

However, the study also identified several limitations. The use of a proportional controller resulted in small steady-state errors, and the ultrasonic sensors introduced measurement uncertainties. Additionally, the mapping process relied on post-processing and did not include real-time localization techniques.

Despite these limitations, the overall performance of the system meets the objectives of the research. The proposed approach provides a practical and efficient solution for autonomous cleaning tasks and serves as a strong foundation for further development.

In conclusion, this work demonstrates that a combination of simple control strategies and sensor-based perception can achieve effective autonomous operation. The system successfully bridges the gap between navigation and mapping, enabling the robot to perform complete area coverage in a structured environment.

7.2 Future Work

Although the proposed autonomous cleaning robot system demonstrates effective performance in simulation, several improvements and extensions can be explored to enhance its capability, accuracy, and real-world applicability.

One of the primary areas for improvement is the control system. The current implementation uses a proportional controller, which introduces steady-state error and limited responsiveness in complex environments. Future work can focus on implementing a PID controller to improve tracking accuracy and reduce oscillations. In addition, adaptive or model-based control strategies could be explored to enable the robot to adjust its behavior dynamically based on environmental conditions.

Another important direction is the enhancement of the sensing system. The current system uses ultrasonic sensors for navigation and a LiDAR model for mapping, but these sensors operate independently. Future work could incorporate sensor fusion techniques to combine data from multiple sensors, resulting in more accurate and reliable perception. This would improve both obstacle detection and environmental understanding.

The mapping approach can also be significantly improved. In the current system, mapping is performed offline using MATLAB after data collection. Future research could focus on implementing real-time mapping techniques such as simultaneous localization and mapping (SLAM). This would allow the robot to build and update the map dynamically while navigating, improving both accuracy and efficiency.

Another potential improvement is the optimization of the cleaning path. The current zigzag coverage strategy provides effective area coverage but does not guarantee optimal efficiency. Advanced path planning algorithms could be implemented to minimize travel distance, reduce overlap, and improve overall performance. Techniques such as grid-based planning, heuristic optimization, or graph-based methods could be considered.

The transition from simulation to real-world implementation is also a critical step for future work. Testing the system on a physical robot would provide valuable insights into real-world challenges such as sensor noise, wheel slip, uneven terrain, and dynamic obstacles. Hardware implementation would also require integration of embedded systems, power management, and communication interfaces.

In addition, the system could be extended to handle more complex and dynamic environments. This includes environments with moving obstacles, irregular boundaries, and varying terrain conditions. Enhancing the robot’s ability to adapt to such conditions would significantly increase its practical applicability.

Finally, future work could focus on improving system autonomy. This includes implementing higher-level decision-making capabilities, such as dynamic path adjustment, environment classification, and task prioritization. Integration with advanced artificial intelligence techniques could further enhance the robot’s ability to operate independently.

7.3 Summary

This chapter has presented a comprehensive summary of the research work, highlighting the key contributions, implementation outcomes, and overall significance of the proposed autonomous cleaning robot system. The study successfully addressed the problem of autonomous environment exploration, mapping, and full-area coverage using a combination of control strategies, sensor integration, and data processing techniques.

The research began with the objective of developing a robot capable of navigating along a curb, understanding its surroundings, and generating a complete cleaning path for the environment. To achieve this, a system architecture was designed that integrates real-time navigation with environmental sensing and offline data processing.

The implementation was carried out using the Pioneer P3DX mobile robot within the CoppeliaSim simulation environment. The robot utilized a differential-drive mechanism controlled through a Lua-based script. A proportional control algorithm was implemented to maintain a desired distance from the curb using ultrasonic sensor feedback. This approach enabled stable and responsive navigation, particularly in structured environments with defined boundaries.

The sensing system played a critical role in achieving the objectives of the study. Ultrasonic sensors were used for real-time obstacle detection and boundary tracking, providing immediate feedback for navigation control. In parallel, a LiDAR sensing system was implemented using vision sensors, allowing the robot to collect detailed spatial data. The LiDAR system generated coordinate-based point cloud data, which formed the basis for mapping.

The collected data was processed in MATLAB, where it was transformed into a structured representation of the environment. Noise reduction and data filtering techniques were applied to improve map clarity. The resulting map accurately represented the geometry of the environment, including boundaries, obstacles, and free space.

Based on the generated map, a cleaning path was created using a hybrid strategy that combines boundary-following and zigzag coverage. This approach ensured that both edge regions and interior areas were effectively covered. The robot successfully executed the generated path, demonstrating its ability to perform systematic area coverage.

The results obtained from the implementation confirmed that the proposed system is capable of achieving autonomous navigation, mapping, and cleaning tasks within a structured environment. The integration of control, sensing, and processing components was shown to be effective and reliable.

In addition to its achievements, the study also identified several limitations. The use of a proportional controller resulted in small steady-state errors, and the ultrasonic sensors introduced measurement uncertainties. The LiDAR implementation, being simulation-based,

does not fully capture real-world sensor behavior. Furthermore, the mapping process relied on offline processing and did not incorporate real-time localization techniques such as SLAM.

Despite these limitations, the system provides a strong foundation for future development. The simplicity of the control strategy, combined with effective sensor integration, demonstrates that complex tasks can be achieved without requiring highly sophisticated algorithms. This makes the system suitable for practical applications where computational resources may be limited.

Overall, this research contributes to the field of autonomous robotics by presenting a complete pipeline from environment exploration to coverage execution. It demonstrates how a robot can autonomously perceive its environment, generate a map, and plan a cleaning path using integrated sensing and control techniques.

The findings of this study highlight the potential for further advancements in autonomous cleaning systems and provide a basis for extending the work into real-world applications. Future improvements in control algorithms, sensor fusion, and mapping techniques can further enhance the performance and applicability of the system.

In conclusion, the developed system successfully achieves the intended objectives and demonstrates the feasibility of autonomous cleaning using a combination of simple yet effective methods. This work lays the groundwork for more advanced and intelligent robotic systems in the future.

8 References

1. Siegwart, R., Nourbakhsh, I. R., & Scaramuzza, D. (2011). *Introduction to Autonomous Mobile Robots* (2nd ed.). MIT Press. https://mitpress.mit.edu/9780262015356/introduction-to-autonomous-mobile-robots/
2. Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic Robotics*. MIT Press. https://mitpress.mit.edu/9780262201629/probabilistic-robotics/
3. Corke, P. (2017). *Robotics, Vision and Control: Fundamental Algorithms in MATLAB* (2nd ed.). Springer. https://link.springer.com/book/10.1007/978-3-319-54413-7
4. Latombe, J.-C. (1991). *Robot Motion Planning*. Springer.
5. Choset, H. (2001). Coverage for robotics – A survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31(1–4), 113–126. https://link.springer.com/article/10.1023/A:1016639210559
6. Galceran, E., & Carreras, M. (2013). A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61(12), 1258–1276. https://www.sciencedirect.com/science/article/pii/S0921889012001880
7. Huang, W. H. (2001). Optimal line-sweep-based decompositions for coverage algorithms. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (pp. 27–32).
8. Zelinsky, A., Jarvis, R. A., Byrne, J. C., & Yuta, S. (1993). Planning paths of complete coverage of an unstructured environment. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (pp. 533–538).
9. Moravec, H. P., & Elfes, A. (1985). High resolution maps from wide angle sonar. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (pp. 116–121).
10. Elfes, A. (1989). Using occupancy grids for mobile robot perception and navigation. *IEEE Computer*, 22(6), 46–57. https://ieeexplore.ieee.org/document/13322

11. Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1), 23–33. https://ieeexplore.ieee.org/document/580977
12. Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1), 90–98.
13. Borenstein, J., Everett, H. R., & Feng, L. (1996). *Navigating Mobile Robots: Systems and Techniques*. AK Peters.
14. Dudek, G., & Jenkin, M. (2010). *Computational Principles of Mobile Robotics* (2nd ed.). Cambridge University Press.
15. Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. (2009). *Robotics: Modelling, Planning and Control*. Springer.
16. Craig, J. J. (2005). *Introduction to Robotics: Mechanics and Control* (3rd ed.). Pearson.
17. Hokuyo Automatic Co., Ltd. (2014). *URG-04LX-UG01 Laser Range Finder Specifications*. https://www.hokuyo-aut.jp/search/single.php?serial=166
18. Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous localization and mapping (SLAM): Part I. *IEEE Robotics & Automation Magazine*, 13(2), 99–110. https://ieeexplore.ieee.org/document/1638022
19. Bailey, T., & Durrant-Whyte, H. (2006). Simultaneous localization and mapping (SLAM): Part II. *IEEE Robotics & Automation Magazine*, 13(3), 108–117. https://ieeexplore.ieee.org/document/1678144
20. Cadena, C., et al. (2016). Past, present, and future of SLAM. *IEEE Transactions on Robotics*, 32(6), 1309–1332. https://ieeexplore.ieee.org/document/7747236
21. LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press. http://planning.cs.uiuc.edu/
22. Ogata, K. (2010). *Modern Control Engineering* (5th ed.). Prentice Hall.
23. Åström, K. J., & Murray, R. M. (2008). *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press. http://www.cds.caltech.edu/~murray/books/AM08/pdf/am08-complete_28Sep12.pdf
24. MathWorks. (2023). MATLAB Documentation. https://www.mathworks.com/help
25. Coppelia Robotics. (2023). CoppeliaSim User Manual. https://www.coppeliarobotics.com/helpFiles/en/index.htm

9 Appendices

9.1 Appendix A

```
sim=require'sim'

function sysCall_init()

    robot=sim.getObject('..')
    motorLeft=sim.getObject("../leftMotor")
    motorRight=sim.getObject("../rightMotor")

    usensors={}
    usensors[1]=sim.getObject("../ultrasonicSensor",{index=3})
    usensors[2]=sim.getObject("../ultrasonicSensor",{index=5})
    usensors[3]=sim.getObject("../ultrasonicSensor",{index=7})

    baseSpeed=2.0
    desiredDist=0.35
    Kp=2.0
    frontDetect=0.40

    startPos=sim.getObjectPosition(robot,-1)

    movedAway=false
end


function readSensor(h)
    local r,d=sim.readProximitySensor(h)
    if r>0 then return d end
    return -1
end


function distanceToStart()
    local pos=sim.getObjectPosition(robot,-1)
    local dx=pos[1]-startPos[1]
    local dy=pos[2]-startPos[2]
    return math.sqrt(dx*dx+dy*dy)
end


function sysCall_actuation()

```

```
40 function sysCall_actuation()
41
42     local dist=distanceToStart()
43     if not movedAway then
44         if dist > 1.0 then
45             movedAway=true
46         end
47     else
48         if dist < 0.5 then
49             sim.setJointTargetVelocity(motorLeft,0)
50             sim.setJointTargetVelocity(motorRight,0)
51             return
52         end
53     end
54
55
56 -- WALL FOLLOW
57
58     local front=readSensor(usensors[1])
59     local frontRight=readSensor(usensors[2])
60     local right=readSensor(usensors[3])
61
62     if (front>0 and front<frontDetect) or
63        (frontRight>0 and frontRight<frontDetect) then
64
65         sim.setJointTargetVelocity(motorLeft,-1.5)
66         sim.setJointTargetVelocity(motorRight,2.5)
67         return
68     end
69
70     if right<0 then
71         sim.setJointTargetVelocity(motorLeft,2.0)
72         sim.setJointTargetVelocity(motorRight,0.5)
73         return
74     end
75
76     local error=desiredDist-right
77     local turn=Kp*error
```

9.2 Appendix B

```
sim=require'sim'

function sysCall_init()
    maxScanDistance=5
    showLines=true
    generateData=true
    rangeData=false -- if false, then X/Y/Z data rel to sensor base
    discardMaxDistPts=true

    self=sim.getObject('..')
    visionSensors={sim.getObject("../sensor1"),sim.getObject("../sensor2")}
    local collection=sim.createCollection(0)
    sim.addItemToCollection(collection,sim.handle_all,-1,0)
    sim.addItemToCollection(collection,sim.handle_tree,self,1)
    sim.setObjectInt32Param(visionSensors[1],sim.visionintparam_entity_to_render,collection)
    sim.setObjectInt32Param(visionSensors[2],sim.visionintparam_entity_to_render,collection)
    sim.setObjectFloatParam(visionSensors[1],sim.visionfloatparam_far_clipping,maxScanDistance)
    sim.setObjectFloatParam(visionSensors[2],sim.visionfloatparam_far_clipping,maxScanDistance)
    red={1,0,0}
    lines=sim.addDrawingObject(sim.drawing_lines,1,0,-1,10000,red)
end

function sysCall_cleanup()
    sim.removeDrawingObject(lines)
end

function sysCall_sensing()
    local measuredData={}

    sim.addDrawingObjectItem(lines,nil)
    for i=1,2,1 do
        local r,t,u=sim.readVisionSensor(visionSensors[i])
        if u then
            local sensorM=sim.getObjectMatrix(visionSensors[i])
            local relRefM=sim.getObjectMatrix(self)
            relRefM=sim.getMatrixInverse(relRefM)
            relRefM=sim.multiplyMatrices(relRefM,sensorM)
            local p={0,0,0}
            p=sim.multiplyVector(sensorM,p)
            t={p[1],p[2],p[3],0,0,0}
            for j=0,u[2]-1,1 do
```

```
            for j=0,u[2]-1,1 do
                for k=0,u[1]-1,1 do
                    local w=2+4*(j*u[1]+k)
                    local v={u[w+1],u[w+2],u[w+3],u[w+4]}
                    if generateData then
                        if rangeData then
                            table.insert(measuredData,v[4])
                        else
                            if v[4]<maxScanDistance*0.9999 or not discardMaxDistPts then
                                p=sim.multiplyVector(relRefM,v)
                                table.insert(measuredData,p[1])
                                table.insert(measuredData,p[2])
                                table.insert(measuredData,p[3])
                            end
                        end
                    end
                    if showLines then
                        p=sim.multiplyVector(sensorM,v)
                        t[4]=p[1]
                        t[5]=p[2]
                        t[6]=p[3]
                        sim.addDrawingObjectItem(lines,t)
                    end
                end
            end
        end
    end
end

```