



Arab Academy for Science, Technology and Maritime Transport

College of Engineering and Technology

Computer Engineering

B. Sc. Final Year Project

**Intelligent Autonomous Disinfection Sterilizing Robot
(iADSRob)**

Presented By:

Jana Ghazy – Youssef Elrefaee – Ahmed Amr – Marwan Alaa Eldin

Supervised By:

Prof. Sherine Yossef – Dr. Essam Seddik

DECLARATION

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Bachelor of Science in Computer Engineering is entirely my own work, that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: Jana Ibrahim Ghazy

Registration No.: 20103705

Date: 1/7/2025.

Signed: Youssef Elreface

Registration No.: 20100326

Date: 1/7/2025.

Signed: Marwan Alaa Eldin

Registration No.: 20100328

Date: 1/7/2025.

Signed: Ahmed Amr

Registration No.: 20103232

Date: 1/7/2025.



ARAB ACADEMY FOR SCIENCE, TECHNOLOGY AND MARITIME TRANSPORT

COLLEGE OF ENGINEERING AND TECHNOLOGY

Department of Computer Engineering

Academic Year: __2024-2025__ Semester: __10__

Senior Project Summary Report

Project Title	<i>Intelligent Autonomous Disinfection Sterilizing Robot (iADSRob)</i>	
Supervisor(s)	<i>Prof. Sherine Yossef – Dr. Essam Seddik</i>	
Team members:	Names Jana Ghazy Youssef Elrefaee Ahmed Amr Marwan Alaa Eldin	Registration Numbers 20103705 20100326 20103232 20100328
Project Deliverables	<ul style="list-style-type: none"> <i>Fully assembled and functional disinfection robot.</i> <i>Android application for tank/battery monitoring.</i> <i>SLAM-based indoor autonomous navigation system.</i> <i>Humidifier-based selective spraying module.</i> <i>System architecture diagram and software flow.</i> <i>Full technical documentation and simulation results.</i> 	
Abstract	<p><i>The iADSRob is an autonomous mobile robot developed to perform intelligent disinfection and sterilization in indoor environments like hospitals. It uses SLAM for navigation, object detection for selective spraying, and a humidifier system to apply disinfectants. A tablet interface displays real-time tank and battery levels. The robot minimizes human exposure, enhances hygiene, and supports healthcare automation.</i></p>	
Engineering Standards	<p><i>IEEE 802.11 (Wi-Fi communication)</i></p> <p><i>IEC 60335-2-98 (Humidifier safety standard)</i></p> <p><i>ROS-compatible messaging and control standards</i></p> <p><i>ISO 13485 (Medical device design guidance, considered for future application)</i></p>	

Design Constraints	<ul style="list-style-type: none"> • <i>Battery life limits continuous operation time.</i> • <i>Must avoid spraying on humans.</i> • <i>Operation confined to flat indoor surfaces.</i> • <i>Limited by weight/size of components.</i> • <i>Budget and availability of safe disinfectant chemicals.</i>
Project Impact	<p><i>Social: Enhances hygiene and public safety.</i></p> <p><i>Economic: Reduces manual labor and hospital costs.</i></p> <p><i>Environmental: Controlled chemical use via selective spraying.</i></p>
Team Organisation	<ul style="list-style-type: none"> ▪ [Jana Ghazy] – app interface, Hardware integration. ▪ [Youssef Elrefaee] – SLAM tuning, humidifier control, battery system. ▪ [Ahmed Amr - Marwan Alaa Eldin] – Object detection, obstacle avoidance, testing & validation, Software development, camera with raspberry pi integration.
Ethics /Safety	<ul style="list-style-type: none"> • <i>Avoids spraying near humans using computer vision.</i> • <i>Ensures safe operation around medical equipment.</i> • <i>Uses non-toxic disinfectant solutions.</i> • <i>Complies with AI fairness and safety principles.</i>

Main Supervisor Signature

.....

ACKNOWLEDGMENT

We would like to express our sincere gratitude to everyone who supported and guided us throughout the development of our graduation project, Intelligent Autonomous Disinfection Sterilizing Robot (iADSRob). First and foremost, we are deeply thankful to our supervisor, [Prof. Sherine Youssef – Dr. Essam Seddik], for their continuous encouragement, technical support, and valuable feedback that significantly contributed to the success of this project.

We also extend our appreciation to the faculty members of the Computer Engineering Department at the Arab Academy for Science, Technology and Maritime Transport for providing us with the essential knowledge, tools, and resources during our academic journey.

A special appreciation goes to our sponsors, ITIDA, whose support and encouragement greatly contributed to making this project a reality.

Special thanks to our families for their unwavering emotional support, patience, and understanding throughout the project timeline. Finally, we are grateful to our colleagues and peers who shared ideas, offered feedback, and collaborated with us during various stages of this work.

ABSTRACT

The Intelligent Autonomous Disinfection Sterilizing Robot (iADSRob) is an innovative solution designed to automate the disinfection process in healthcare facilities and public environments, aiming to mitigate the spread of infectious diseases. In response to the global demand for efficient and contactless sanitation technologies, iADSRob integrates autonomous navigation, intelligent path planning, and object detection to target high-contact surfaces such as door handles, hospital beds, equipment, and workspaces. The robot is equipped with a LiDAR-based SLAM (Simultaneous Localization and Mapping) system for real-time environment mapping and localization, allowing it to autonomously navigate complex indoor environments while avoiding obstacles. A vision-based module enables selective spraying to avoid humans and target only relevant surfaces. The system includes two tanks for disinfection and sterilization liquids, with real-time level monitoring displayed on a mounted tablet via a custom-built Android application. Powered by a Raspberry Pi microcontroller, the robot uses a combination of sensor inputs, motor drivers, and peristaltic pumps to control movement and disinfection functions. The project also emphasizes user safety, chemical control, and operational efficiency. This report details the full design process of the robot, from conceptualization and mechanical assembly to software development and testing. It also includes a review of existing disinfection systems, the justification for using autonomous and intelligent methods, system architecture, implementation constraints, performance evaluation, and a business model outlining market potential. iADSRob demonstrates a practical implementation of smart robotics in healthcare and sanitation, contributing to safer, cleaner environments with minimal human intervention.

TABLE OF CONTENTS

List of figures	iii
list of tables	v
List of acronyms/abbreviations	vi
1 Introduction	1
1.1 Overview	1
1.2 Motivation and Applications	1
1.3 Challenges	2
1.4 Problem Statement	2
1.5 Objective	2
1.6 Thesis Outline.....	3
2 Literature Review and Related Work.....	4
2.1 Existing Autonomous Disinfection Robots	4
2.2 Navigation and Localization Techniques	6
2.3 Selective Disinfection and Safety	6
2.4 Summary and Gap Analysis	7
3 Project Terminology	8
3.1 SLAM (Simultaneous Localization and Mapping)	8
3.2 Raspberry Pi	8
3.3 LiDAR (Light Detection and Ranging)	8
3.4 Humidifier	8
3.5 Disinfection vs. Sterilization	8
3.6 Selective Spraying	9
3.7 Android Application	9
3.8 Obstacle Avoidance	9
3.9 Path Planning.....	9
3.10 Autonomous Navigation.....	9
4 Proposed Model.....	10
4.1 System Objectives and constraints.....	10
4.2 System prototype & synthesis.....	11
4.3 System architecture Diagram.....	22
4.4 Software Architecture and Middleware.....	24
4.5 AI Integration and Object Detection Model	42
4.6 Model Implementation and Code Workflow	45
4.7 Kinect-Based RGB Input Integration	48
4.8 description of each phase.....	51
4.9 Design Issues & limitation	52
4.10 Safety Design and Power Management.....	53
4.11 Dual-Mode Operation: Autonomous and Manual Control	54
5 Project Simulation and Performance Evaluation	56
5.1 Mapping and Localization with SLAM Toolbox	56
5.2 Navigation with ROS2 Nav2 Stack.....	58
5.3 Simulation and Visualization: Gazebo and RViz2.....	61
5.4 AI Model Performance Evaluation	64
5.5 Real-World Testing and Deployment.....	65
5.6 Kinect-Based Technical Enhancements	65
Kinect RGB-D Capture:	66
YOLOv8 Inference Pipeline:	67
6 Business Model.....	68
6.1 Business Model Canvas.....	68
6.2 Components of the Business Model	69
7 Conclusion and Future Work.....	72
7.1 Conclusion.....	72
7.2 Future Enhancements	73

References	75
Appendix A: Formatting Description.....	Error! Bookmark not defined.
Appendix B: General Recommendations	Error! Bookmark not defined.
Appendix C: Citation and Referencing	Error! Bookmark not defined.

LIST OF FIGURES

Table 1: Comparison of Existing Autonomous Disinfection Robot Approaches and Their Limitations.....	5
Figure 1.3: Completed Frame with Wheel Integration.....	11
Figure 1.1: Base Frame of the Robot Structure	11
Figure 1.2: Assembling the Vertical Supports.....	11
Figure 1.5:Printed Bracket Close-Up	13
Figure 1.4: 3D printing components of the robot using FDM technology.	13
Figure 1.6: Mounting the 3D-printed components onto the aluminum frame.	13
Figure 1.7: Completed robot prototype with mounted disinfection sprayer	15
Figure 1.8: Back view of iADSRob showing the dual tank compartment	15
Figure 1.9: Tablet-based software interface displaying tank levels and battery monitoring	16
Figure 2: Motor control circuit using an L298N driver and Arduino, enabling directional and speed control of the robot’s dual DC motors.	19
Figure 2.1: System Architecture Diagram of iADSRob Robot	22
Figure 2.2: <i>Example of a mobile robot platform used to demonstrate ROS2 architecture with mounted LiDAR sensor and modular layer structure for perception, planning, and control.</i>	26
Figure 2.3: <i>: ROS2 Control Architecture showing the layered interaction between controllers, interfaces, and hardware components.</i>	28
Figure 2.4: ROS2 Humble Hawksbill – the chosen LTS distribution for this project, ensuring stability and compatibility with Ubuntu 22.04.	33
Figure 2.5: <i>Raspberry Pi setup used for running the robot's operating system and managing network communication through USB and Ethernet connections.</i>	35
Figure 2.6: <i>Bluetooth control interface used to send serial commands from the mobile device to the robot via the HC-05 module for manual operation.</i>	41
Figure 2.7: GPU environment setup using NVIDIA Tesla T4, confirming CUDA compatibility for training YOLOv8 models with the annotated dataset.	45
Figure 2.8: YOLOv8s training command configuration using a custom dataset with 500 epochs, 500×500 image size, and enabled augmentation and plotting features.	46
Figure 2.9: Confusion matrix visualization showing the model’s classification performance across object classes after training.	47
Figure 3.1: Training performance metrics plotted during YOLOv8 model training, including box loss, classification loss, DFL loss, precision, recall, and mean average precision (mAP) scores.	47

Figure 5: Installation and configuration steps for libfreenect and udev rules to enable Kinect device access on Linux systems.....	49
Figure 5: Custom udev rules assigning read/write permissions for Kinect motor, audio, and camera USB interfaces using vendor ID 045e and respective product IDs.	49
Figure 6: Commands used to reload and apply udev rules, ensuring immediate activation of new USB device permissions.....	49
Figure 7: <i>Virtual environment setup and package installation script for integrating YOLOv8 with Kinect using NumPy, OpenCV, and freenect bindings</i>	50
Figure 3.2: Generated 2D occupancy grid map using SLAM Toolbox, enabling real-time mapping and localization for autonomous indoor navigation.....	56
Figure 3.3: ROS2 Nav2 Stack in action, showing global and local path planning for autonomous navigation within a mapped environment.	58
Figure 3.4: Gazebo and RViz2 simulation environments used to test robot behavior, obstacle avoidance, and sensor perception before real-world deployment.	61
Table 3.....	64
Figure 8: Code snippet demonstrating YOLOv8 inference configuration optimized for real-time performance on Raspberry Pi using reduced image size and FP16 precision	67
Table 4.....	68
Figure 0-1: Page settings.	Error! Bookmark not defined.
Figure 0-2: Paragraph settings.	Error! Bookmark not defined.
Figure 0-3: Setting caption numbering to include chapter number.	Error! Bookmark not defined.
Figure 0-4: Using Cross-reference.	Error! Bookmark not defined.

LIST OF TABLES

Table 0-1: List of headings and their formatting.	Error! Bookmark not defined.
--	-------------------------------------

LIST OF ACRONYMS/ABBREVIATIONS

ACRONYM	Definition of Acronym
AI	Artificial Intelligence
RGB	Red, Green, Blue
YOLO	You Only Look Once
SLAM	Simultaneous Localization and Mapping
ROS	Robot Operating System
CPU	Central Processing Unit
GPU	Graphics Processing Unit
RPi	Raspberry Pi
OpenCV	Open Source Computer Vision Library
BGR	Blue, Green, Red (OpenCV image format)
FP16	16-bit Floating Point Precision
mAP	Mean Average Precision
API	Application Programming Interface
ABI	Application Binary Interface

Chapter One

1 INTRODUCTION

1.1 OVERVIEW

In light of recent global health crises, the importance of effective and efficient disinfection systems has become more evident than ever. The Intelligent Autonomous Disinfection Sterilizing Robot (iADSRob) is a mobile robotic platform developed to autonomously navigate indoor environments and disinfect high-contact surfaces using selective spraying mechanisms. It integrates intelligent decision-making, obstacle avoidance, and real-time mapping using SLAM (Simultaneous Localization and Mapping) technology, supported by a robust control system based on Raspberry Pi.

1.2 MOTIVATION AND APPLICATIONS

Manual disinfection processes are time-consuming, labor-intensive, and pose a risk to human operators. The outbreak of pandemics like COVID-19 revealed the urgent need for automated, intelligent solutions in healthcare and public spaces. iADSRob addresses this need by providing a reliable, autonomous system that reduces human exposure to pathogens and enhances disinfection efficiency.

Its applications include hospitals, intensive care units (ICUs), airports, schools, public transportation stations, and offices—anywhere frequent and safe surface disinfection is required.

1.3 CHALLENGES

Developing iADSRob involved addressing several technical and operational challenges, including:

- Accurate mapping and localization in indoor environments using SLAM.
- Designing a selective spraying system that avoids spraying on humans.
- Ensuring real-time communication between hardware and a user interface.
- Power management across multiple components (motors, pumps, sensors, tablet).
- Navigating complex environments without collisions.

1.4 PROBLEM STATEMENT

There is a growing demand for intelligent, contactless disinfection systems that can autonomously operate in complex indoor environments. Traditional methods fall short in terms of safety, consistency, and efficiency. The problem lies in developing a robotic system that can identify priority areas for disinfection, avoid humans, and execute its tasks with minimal human intervention.

1.5 OBJECTIVE

The main objective of this project is to design and implement a fully autonomous mobile robot capable of:

1. Mapping and navigating indoor environments.
2. Identifying and targeting high-contact surfaces.
3. Disinfecting and sterilizing these surfaces selectively.
4. Monitoring tank levels and battery status via a user-friendly interface.

1.6 THESIS OUTLINE

This report is structured as follows:

- Chapter Two reviews existing technologies and related work in autonomous disinfection.
- Chapter Three defines key terminology and concepts used throughout the project.
- Chapter Four details the proposed model, including hardware, software, system architecture, and limitations.
- Chapter Five presents the simulation results and performance evaluation of the system.
- Chapter Six outlines the business model and market potential.
- Chapter Seven concludes the report and discusses possible future enhancements.

2 LITERATURE REVIEW AND RELATED WORK

The emergence of global pandemics has highlighted the urgent need for innovative disinfection technologies that reduce human contact and increase efficiency. In recent years, a variety of autonomous robots have been developed for cleaning, sterilizing, and disinfecting purposes in hospitals, public buildings, and transport facilities. This chapter reviews the most relevant prior work and technologies related to autonomous disinfection robots, including navigation systems, disinfection methods, and safety protocols.

2.1 EXISTING AUTONOMOUS DISINFECTION ROBOTS

Several commercial and research-based robots have been designed to automate disinfection using methods such as ultraviolet (UV-C) light, chemical spraying, or steam. For example:

1. **Xenex LightStrike**, **UVD Robots**, and **Tru-D SmartUVC** use (UV-C) light to disinfect hospital rooms, offering effective pathogen elimination but facing limitations in shadowed or occluded areas and typically commanding high costs [11], [12], [13].
2. A notable advancement is the smart robot deployed in the **Masjid al-Harām (Grand Mosque)** in Makkah. This autonomous SLAM-based robot disinfects large surface areas and distributes **Zamzem water bottles** without human intervention. It operates via a pre-programmed map across six levels, covers about **600 m²** per round, runs **5–8 hours** on battery, and uses fine dry fog to optimize sterilization [14], [15].
3. In industrial and outdoor settings, robots like the **S8.2-TF thermal fog robot** offer autonomous, long-range aerosol disinfection over kilometers of outdoor space, equipped with GPS, visual navigation, and obstacle avoidance [16].

4. Academic research includes robots like UltraBot and vision-based UAV systems:

- **UltraBot** combines physical wiping and UV-C disinfection to effectively reduce hospital surface bacterial counts by over 90% in real-world settings [17].
- A UAV-based disinfecting system selectively sprays high-touch surfaces like door handles in indoor environments using vision-based targeting [18].

Table 1: Comparison of Existing Autonomous Disinfection Robot Approaches and Their Limitations

Approach	Strengths	Limitations
1. UV-C Robots (Xenex, UVD, Tru-D)	Proven effectiveness, widely used in hospitals	Poor coverage of shaded areas, high cost
2. Haram SLAM Robot	Large coverage, intelligent fog dispensing	Specific to mosque environment
3. Industrial Thermal Robots	Long-range, autonomous outdoor operation	Not suited for indoor settings
4. UltraBot, UAV Vision Robots	Targeted cleaning, combines UV + mechanical	Mostly in research phase, limited deployment

While effective, many of these robots—particularly those using UV-C light—struggle to disinfect shaded or occluded surfaces and are often limited to sterile environments like hospitals. Although advanced models like the Harām Mosque robot demonstrate the integration of chemical spraying and SLAM navigation for wide-area coverage, such systems are often tailored to specific facilities and can be costly to scale. As a result, there remains a need for more versatile, affordable, and selective disinfection solutions adaptable to diverse indoor environments.

2.2 NAVIGATION AND LOCALIZATION TECHNIQUES

Navigation in autonomous robots commonly uses SLAM (Simultaneous Localization and Mapping), which allows the robot to build a map of its environment while tracking its own position within it. SLAM is particularly useful in indoor environments where GPS is not available.

Robots like TurtleBot, ClearPath Husky, and MiR100 leverage SLAM-based systems combined with LIDAR and camera sensors to navigate hospitals and warehouses. In addition, path planning algorithms such as A*, Dijkstra, and RRT (Rapidly Exploring Random Tree) are often employed to compute optimal collision-free routes.

2.3 SELECTIVE DISINFECTION AND SAFETY

A key limitation in existing robots is their lack of selective targeting. Most systems apply disinfection uniformly, which can waste resources and inadvertently expose humans or sensitive equipment. Recent research proposes integrating computer vision and object detection models to identify high-contact surfaces like door handles, elevator buttons, or hospital beds.

Some projects use YOLO (You Only Look Once) or Faster R-CNN models to detect humans and avoid spraying directly on them. Others use depth sensors and infrared imaging to improve detection in low-light environments.

2.4 SUMMARY AND GAP ANALYSIS

Although many advancements have been made in autonomous disinfection, existing solutions often:

- Lack selective disinfection capabilities.
- Do not provide real-time monitoring of disinfection fluid levels.
- Are prohibitively expensive or designed for specific environments.
- Use UV light, which cannot reach shaded surfaces.

The **iADSRob project** addresses these gaps by combining low-cost hardware, real-time SLAM navigation, selective spraying based on surface recognition, and a user interface for tank and battery monitoring. It offers a more accessible, safer, and intelligent alternative to existing robotic disinfecting systems.

3 PROJECT TERMINOLOGY

3.1 SLAM (SIMULTANEOUS LOCALIZATION AND MAPPING)

SLAM is a technique used by autonomous robots and vehicles to build a map of an unknown environment while simultaneously keeping track of their location within it. It is essential for indoor navigation where GPS is not available.

3.2 RASPBERRY PI

A low-cost, credit-card-sized computer used as the main microcontroller in the robot. It processes sensor data, controls hardware components, and communicates with the user interface.

3.3 LIDAR (LIGHT DETECTION AND RANGING)

LiDAR is a remote sensing method that uses laser pulses to measure distances to objects. It enables the robot to detect obstacles and generate accurate maps for navigation.

3.4 HUMIDIFIER

A humidifier is used in the robot to disperse disinfectant or sterilizing mist into the air. Unlike traditional pump-based spraying systems, the humidifier produces fine particles through ultrasonic or thermal vibration methods, allowing for uniform, contactless disinfection. This approach is quieter, energy-efficient, and safer for use around sensitive equipment and humans.

3.5 DISINFECTION VS. STERILIZATION

- Disinfection: The process of eliminating most pathogenic microorganisms (except bacterial spores) on surfaces.

- Sterilization: A higher-level process that destroys all forms of microbial life, including spores.

3.6 SELECTIVE SPRAYING

A system feature that enables the robot to apply disinfectant only to high-contact or contaminated surfaces, avoiding spraying on humans or irrelevant areas. This is achieved through object detection and path planning.

3.7 ANDROID APPLICATION

A custom-built mobile app that displays real-time status of the robot, including tank levels, battery levels, and system messages. It provides a user-friendly interface for hospital staff or operators.

3.8 OBSTACLE AVOIDANCE

A safety mechanism that allows the robot to detect and avoid collisions using real-time sensor feedback from LiDAR, ultrasonic sensors, or vision systems.

3.9 PATH PLANNING

The algorithmic process of determining the best route for the robot to take to cover an area efficiently while avoiding obstacles and minimizing energy use.

3.10 AUTONOMOUS NAVIGATION

Refers to the robot's ability to move and make decisions without human input. It combines SLAM, obstacle avoidance, and path planning to enable full autonomy.

4 PROPOSED MODEL

4.1 SYSTEM OBJECTIVES AND CONSTRAINTS

The main objectives of the iADSRob project are:

1. To design and build an autonomous mobile robot capable of navigating indoor environments using **SLAM**.
2. To develop a **humidifier**-based disinfection system that can be activated selectively.
3. To integrate a **computer vision** module to detect and avoid humans or irrelevant targets during spraying.
4. To create a user interface (**Android app**) that displays real-time tank and battery levels.
5. To enhance hygiene efficiency in hospitals and public buildings with minimal human involvement.

During development, the team faced several realistic constraints, including:

- Power limitations: The system depends on multiple battery sources (Raspberry Pi, tablet, motor), requiring careful power management.
- Indoor-only operation: The robot is limited to flat, indoor environments due to SLAM and hardware limits.
- Component size and weight: The design had to remain compact and lightweight for safe movement and stability.
- Budget limitations: The team had to choose affordable components and avoid high-cost hardware like industrial LiDAR or UV lamps.
- Safety restrictions: Spraying had to avoid humans, sensitive electronics, and be chemically safe.
- Sensor accuracy: Obstacle detection and SLAM were limited by affordable sensor precision and environmental noise.

4.2 SYSTEM PROTOTYPE & SYNTHESIS

4.2.1 Assembly & Integration



Figure 1.1: Base Frame of the Robot Structure



Figure 1.2: Assembling the Vertical Supports



Figure 1.3: Completed Frame with Wheel Integration

The construction of the iADSRob prototype began with the mechanical assembly of the structural frame, which serves as the foundation for all subsystems including electronics, disinfection mechanisms, and the user interface. The following steps were followed to ensure a robust and modular design:

1. Base Frame Construction

The foundation of the robot was built using lightweight aluminium extrusion profiles, providing both strength and modularity. This base frame was designed to support the tanks, electronic components, and mechanical actuators. It also includes mounting space for the drive system and caster wheels for enhanced mobility.

Figure 1.1 illustrates the completed base frame prior to any vertical attachments.

2. Mounting the Vertical Supports

Vertical aluminium columns were then fixed to the base frame to provide height for mounting the sprayer assembly and user interface. These supports are critical to organizing the layout of the internal components, such as electronics, wiring channels, and structural reinforcements. As shown in Figure 1.2

3. Wheel Integration and Mobility Setup

After constructing the main frame, two rear wheels were integrated to enable differential drive using a hoverboard motor, while front caster wheels ensured maneuverability and balance. The motorized wheels were selected for their torque capacity and compatibility with the Raspberry Pi control system.

Figure 1.3 demonstrates the completed mechanical frame with the full set of wheels installed, forming the mobile platform base.

This structural integration laid the groundwork for further development, such as the installation of 3D-printed components, tanks, disinfection nozzles, sensors, and the control tablet.

4.2.2 3D Printing Process – Fabrication of Robot Parts

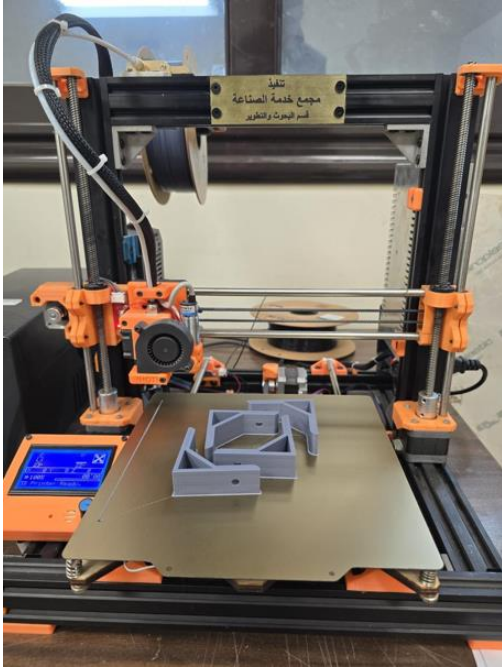


Figure 1.4: 3D printing components of the robot using FDM technology.



Figure 1.5: Printed Bracket Close-Up



Figure 1.6: Mounting the 3D-printed components onto the aluminum frame.

To support the mechanical and functional needs of the iADSRob, a number of **custom components** were designed and fabricated using **Fused Filament Fabrication (FFF) 3D printing**. This approach allowed the team to create precise, lightweight, and cost-effective parts tailored to the robot's requirements.

1. Printing of Custom Mechanical Parts

Several robot parts, including brackets, mounts, and holders for electronic components, were modeled using CAD software and printed using PLA filament. This process provided rapid prototyping capabilities and ensured mechanical compatibility with the aluminium extrusion frame.

Figure 1.4 shows the active printing process of a part on the 3D printer using FFF technology.

2. Bracket Design and Mounting

A variety of custom brackets were printed to securely attach components such as the ultrasonic sensors, tablet holder, and camera mount to the robot's frame. These brackets were optimized for strength, fit, and ease of integration.

Figure 1.5 provides a close-up view of one of the printed brackets, while Figure 1.6 shows the mounting process of these parts onto the aluminium structure.

The 3D-printed elements played a vital role in enabling customization, modular assembly, and precise alignment of critical parts, helping achieve a professional and functional robotic design.

4.2.3 Final Prototype of the Robot



Figure 1.7: Completed robot prototype with mounted disinfection sprayer



Figure 1.8: Back view of iADSRob showing the dual tank compartment

After the frame construction, part fabrication, and system integration were completed, the final prototype of the iADSRob was assembled. This prototype brought together all mechanical, electrical, and software components into a cohesive, functional unit.

1. Complete External Structure

The final design features a tall, vertical aluminum frame with a solid base housing the robot's electronics, power system, and tank compartments. The upper frame holds the disinfection spraying system and the tablet interface used for real-time system monitoring. For environmental awareness and autonomous navigation, the robot is equipped with multiple sensors: a depth camera mounted below the sprayer for detecting obstacles and identifying targets, a LiDAR sensor positioned at the lower front of the base to map the environment and support SLAM-based navigation, and ultrasonic sensors distributed around the body to assist in short-range obstacle detection and collision avoidance. Figure 1.7 shows the completed iADSRob prototype with the disinfection sprayer mounted at the center and the display screen attached to the top frame.

2. Tank Compartment Integration

The robot includes a hidden compartment that securely holds two separate tanks: one for disinfection liquid and the other for sterilization liquid. These tanks are safely enclosed and connected to the humidifier spraying mechanism. Figure 1.8 illustrates the back view of the robot with the compartment opened, revealing the two mounted tanks, each fitted with its respective tubing and wiring for pump control.

This final prototype was tested and presented successfully, demonstrating autonomous movement, obstacle detection, spraying capabilities, and real-time system monitoring via the Android tablet.

4.2.4 Software Interface

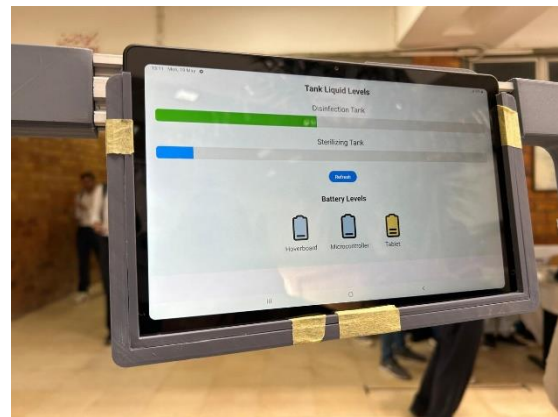


Figure 1.9: Tablet-based software interface displaying tank levels and battery monitoring

The iADSRob features a dedicated Android tablet interface mounted at the top of the robot frame, serving as the primary means of system monitoring and interaction. The software displays real-time information about the liquid levels in both the **Disinfection Tank** and **Sterilizing Tank**, represented by dynamic progress bars that update upon user request. Additionally, it shows the battery status for the three key components: **hoverboard motor**, **microcontroller (Raspberry Pi)**, and the **tablet itself**, each indicated by distinct battery icons. A refresh button allows users to manually update the displayed values. This user-friendly interface ensures that hospital staff or maintenance personnel can quickly assess system readiness and resource levels at a glance, enhancing the robot's practicality and usability in real-world healthcare environments.

4.2.5 Hardware Architecture and Integration

The hardware architecture of the disinfection robot is logically structured into two complementary layers: a high-level processing unit and a low-level control unit. This separation is essential for achieving real-time responsiveness, modularity, and robustness across the system. At the core of the high-level layer is the Raspberry Pi 4, a powerful single-board computer responsible for executing the core ROS2 stack. It handles high-level computational tasks such as SLAM (Simultaneous Localization and Mapping), path planning, sensor data processing, and overall system coordination. The Pi also manages Bluetooth communication and serves as the central node orchestrating data flow across subsystems.

The low-level control layer is managed by an Arduino Mega 2560, which acts as the interface between the software architecture and the robot's physical actuators and sensors. The Arduino receives velocity commands and actuator instructions from the Raspberry Pi via serial communication. It translates these commands into electrical signals that drive the motors and relays. This division allows the Raspberry Pi to remain focused on time-sensitive decision-making and sensor fusion, while the Arduino provides reliable, low-latency interaction with hardware-level components.

The DC motors are controlled via dedicated motor drivers connected to the Arduino. The motors are equipped with encoders to enable velocity tracking and precise movement control. The Arduino reads encoder signals and applies feedback-based corrections to ensure accurate trajectory execution. Additionally, two relays are interfaced with the Arduino to control the on/off state of the humidification units—one dedicated to disinfection and another to perfuming. These relays respond instantly to serial commands issued by the Pi, facilitating synchronized actuation within planned navigation routines.

For environmental perception, the robot utilizes the RPLiDAR A1, a 2D laser scanner connected via USB to the Raspberry Pi. The LiDAR provides real-time 360-degree scans with a range of 6–12 meters and a scanning frequency of 5–10 Hz. These scans are published to ROS2 topics and used by both the SLAM Toolbox for map generation and by the Nav2 stack for obstacle avoidance and dynamic path planning. Its resolution and range make it particularly suited for structured indoor environments such as hospital corridors, classrooms, and commercial buildings.

An additional layer of operational flexibility is introduced through the Bluetooth HC-05 module, which is connected to the Arduino. This module enables manual control via a smartphone application, allowing operators to send motion commands or toggle the humidifiers remotely. The system uses a lightweight serial protocol for this communication, with predefined command sets that ensure reliable parsing and execution. This functionality acts as a safety override or fallback mechanism when autonomous behavior is undesirable or not feasible.

The combination of a hybrid processing model (Raspberry Pi + Arduino), modular sensor-actuator interfacing, and a dual-mode control strategy (autonomous + Bluetooth) ensures that the robot can perform robustly across a range of environments and scenarios. This architecture not only improves fault isolation and ease of debugging but also makes the platform extensible for future enhancements such as additional sensors, camera modules, or advanced actuator systems.

4.2.6 Kinect v1 Camera Integration for Object Detection

Although not essential for the robot’s core disinfection functionality, the integration of a Kinect v1 RGB-D sensor adds valuable capabilities for object detection and contextual awareness in complex environments. The Kinect v1 provides synchronized RGB images and depth maps, enabling the robot to detect and classify objects within its surroundings using spatial cues. When mounted on the disinfection robot, the Kinect connects to the Raspberry Pi via USB and is operated using ROS2-compatible drivers such as `iai_kinect2` or alternative wrappers designed to support legacy Kinect hardware. Once initialized, the sensor streams data through standardized ROS2 topics, including `/camera/rgb/image_raw`, `/camera/depth/image_raw`, and `/camera/depth/points`.

These image and depth streams can be processed by onboard or external object detection nodes employing classical computer vision or machine learning methods. For instance, YOLO-based ROS2 packages or OpenCV-based detection pipelines can subscribe to the RGB and depth data to identify specific objects—such as obstacles, furniture, or signage—and infer their distance and orientation from the robot. This depth-aware object recognition enables more intelligent decision-making, such as rerouting around dynamic objects, verifying that disinfection was performed near critical assets, or detecting human presence in sensitive areas.

The data can also be visualized in RViz2 for debugging and monitoring purposes. Real-time overlays of detected object bounding boxes, depth annotations, and camera perspectives aid in verifying algorithm performance during development. To mitigate the high computational load associated with depth data processing, the Kinect module is activated only when object detection is required. In resource-constrained setups, the RGB-D processing can be offloaded to a dedicated vision co-processor or companion system, preserving the real-time performance of the navigation and disinfection subsystems.

This optional integration of the Kinect sensor transforms the robot into a more perceptive platform capable of spatial reasoning and context-sensitive operation, laying the foundation for advanced features such as semantic SLAM, targeted disinfection, or interaction with dynamic environments.

4.2.7 Motor Driver and Motion Control Logic

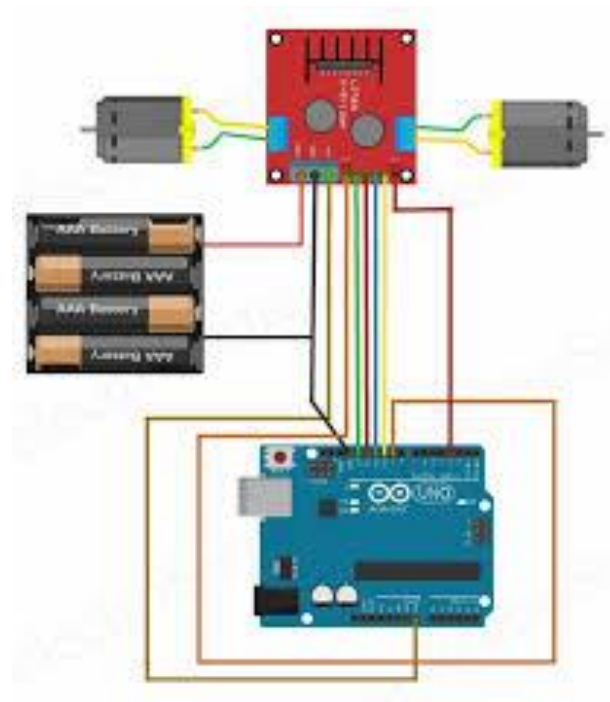


Figure 2: Motor control circuit using an L298N driver and Arduino, enabling directional and speed control of the robot's dual DC motors.

The robot is equipped with two DC motors connected to a motor driver module, typically an L298N or similar dual H-Bridge controller. The motor driver is responsible for converting directional and speed commands into high-current voltage outputs suitable for

driving the motors. This setup provides bidirectional control over each motor, enabling the robot to move forward, backward, and perform pivoting turns.

The Arduino Mega receives motion commands from the Raspberry Pi over the serial interface. These commands are sent in simple text format—such as "FWD", "BACK", "LEFT", "RIGHT", and "STOP"—and are interpreted in real time by the Arduino firmware. Upon receiving a command, the Arduino triggers specific control logic that activates the appropriate EN (enable) and IN1/IN2 pins of the motor driver. This logic dictates the rotation direction and speed of each motor, allowing the robot to execute precise maneuvers based on input from the higher-level navigation system or Bluetooth manual control.

For smooth and adjustable motion, the control logic employs PWM-based speed control. PWM (Pulse Width Modulation) signals are generated using the `analogWrite()` function on the Arduino and are applied to the enable pins of the motor driver. This allows dynamic velocity modulation depending on the navigation context—whether cautious, slow movement is needed in tight spaces or faster motion is required in open areas. Speed settings can be tuned via ROS parameters or overridden manually through Bluetooth commands issued by the user.

To improve safety and fault tolerance, the Arduino firmware incorporates a timeout mechanism that automatically stops the motors if no new movement command is received within a predefined time window. This safety feature ensures that the robot does not continue moving in the event of a communication failure, system crash, or unresponsive software component. When the timeout is triggered, all PWM signals are halted, and motor driver inputs are disabled, bringing the robot to a safe stop.

This motor control system balances responsiveness, precision, and safety, forming a reliable foundation for both autonomous navigation and manual intervention.

4.2.8 Relay Logic and Humidifier Control System

The disinfection robot integrates two dedicated humidifiers, each assigned a distinct operational role—one for disinfection using sterilizing chemical solutions, and the other

for perfuming to enhance the ambient environment. These devices are controlled via two independent relay modules, which serve as electrically isolated switches driven by digital output pins on the Arduino Mega. The separation of functions enables the system to alternate between sterilization and aromatization tasks or run both routines simultaneously, depending on the mission requirements.

Each relay is connected to a digital pin of the Arduino and powered through a stable 5V supply, typically regulated through a buck converter or onboard regulator. The relays themselves control either 220V AC or 12V DC humidifiers, depending on the power specifications of the devices in use. Because relays act as high-voltage switches triggered by low-power logic signals, they offer a safe and effective means of integrating household or industrial-grade appliances with microcontroller-based control systems. In their default state, the relays are configured as Normally Open (NO), ensuring that the humidifiers remain inactive unless explicitly triggered by control logic.

The command handling for these humidifiers is embedded within the Bluetooth serial communication routine on the Arduino. When the user sends a command from the mobile interface—for instance, "H1_ON" to activate the disinfection humidifier or "H2_OFF" to deactivate the perfume unit—the Arduino parses the command string and toggles the relevant digital pin to a HIGH or LOW state. This state change energizes or deactivates the relay coil, thereby switching the humidifier ON or OFF. The modular nature of the command protocol supports precise manual control, allowing users to initiate specific routines based on real-time observations or needs.

To safeguard the hardware against misuse, overheating, or power-related stress, the Arduino firmware includes internal timers that limit the activation period of each humidifier. For example, a single run cycle may be capped at three to five minutes to prevent excessive operation. This time constraint is enforced programmatically, after which the relay is automatically deactivated regardless of additional incoming commands. Furthermore, debounce logic is employed to avoid accidental multi-triggering due to communication noise or button press jitter, enhancing overall system stability.

To provide user feedback and operational confidence, the system optionally transmits acknowledgment messages via Bluetooth after each command is executed. These acknowledgments can be displayed on the user's mobile interface to confirm that the

requested action—such as humidifier activation or deactivation—was successfully carried out. By combining physical relay control with software safeguards and intuitive manual commands, the robot delivers a flexible, safe, and user-friendly mechanism for managing environmental treatment functions in real-time.

4.3 SYSTEM ARCHITECTURE DIAGRAM

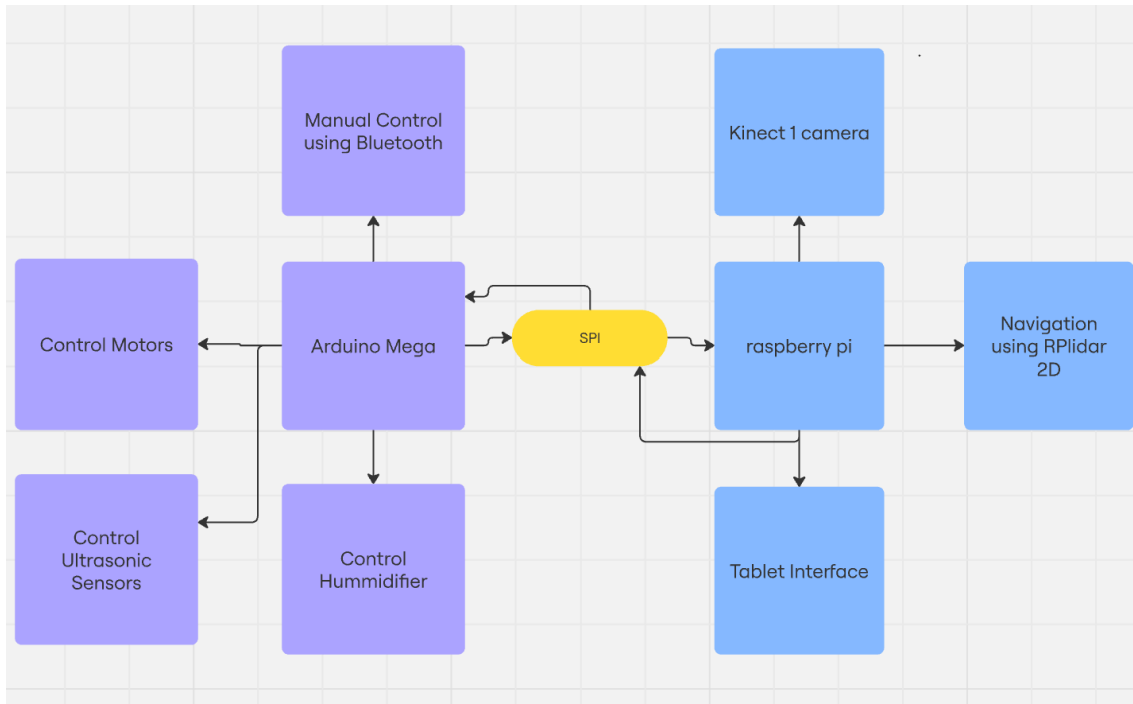


Figure 2.1: System Architecture Diagram of iADSRob Robot

Figure 2.1 The overall architecture of the Intelligent Autonomous Disinfection Sterilizing Robot is designed around a modular hardware-software separation that ensures robustness, expandability, and fault isolation. The hardware layer is composed primarily of two main processing units: a Raspberry Pi 4 and an Arduino Mega. The Raspberry Pi 4 functions as the central processing unit for high-level tasks. It runs ROS2 Humble and hosts the full navigation stack, including the SLAM Toolbox for map generation and the Nav2 stack for localization, path planning, and control. It also interfaces with the RPLiDAR A1 sensor, which continuously streams 2D laser scans used for mapping and real-time obstacle detection. Optionally, a Kinect v1 camera can be connected to the Pi to provide RGB-D data for future extensions involving obstacle avoidance or semantic

perception. Additionally, the Raspberry Pi communicates with a tablet interface, which serves as the user-facing dashboard for remote control and monitoring.

On the low-level control side, the Arduino Mega is tasked with direct interaction with actuators and simple sensors. It receives command signals from the Raspberry Pi via an SPI interface, interpreting those commands to control the motor driver (L298N) and the relay modules responsible for switching the two onboard humidifiers. It also manages the HC-05 Bluetooth module, which receives manual control commands in scenarios where wireless override is necessary. In systems equipped with ultrasonic sensors, the Arduino is responsible for reading and processing proximity data to support collision avoidance during manual navigation.

From a software perspective, the ROS2 architecture follows a standard node-based design. The `rplidar_node` publishes laser scan data to the `/scan` topic, which is consumed by either the `slam_toolbox` node during mapping or the `amcl` node during localization. The navigation stack includes `map_server`, `planner_server`, and `controller_server`, each of which is responsible for maintaining the static map, generating global and local plans, and issuing velocity commands to the robot. These velocity commands are published to the `/cmd_vel` topic, which is then read by a serial interface node and transmitted to the Arduino. The Arduino translates this data into PWM signals that drive the motors. In cases where manual override is enabled, a custom `bluetooth_listener_node` may be launched. This node listens for ASCII commands sent via Bluetooth and transmits them directly to the Arduino for immediate execution.

The data exchange between Raspberry Pi and Arduino via SPI ensures low-latency communication, critical for maintaining smooth real-time control. Additionally, the use of ROS2 transforms (TF) manages spatial relationships among the robot's coordinate frames, enabling consistent and accurate localization in dynamic environments.

This system architecture supports flexible operation in multiple modes. In autonomous mode, the robot performs SLAM-based navigation to predefined waypoints with minimal user interaction. In manual mode, the operator can assume full control via Bluetooth. The hybrid mode allows for mid-operation intervention, where the robot can switch between automatic and manual control seamlessly. This flexibility is essential for deployment in real-world indoor environments where static maps may not always be reliable and human

oversight remains necessary. The architecture also ensures that in the event of a software failure on the Raspberry Pi, the Arduino can continue to respond to Bluetooth commands, allowing for recovery and repositioning without requiring a full system reboot.

4.4 SOFTWARE ARCHITECTURE AND MIDDLEWARE

As implemented in our iADSRob robot, the ROS2 node-based architecture supports modular processing and real-time data integration. Each node is responsible for a specific function like navigation, mapping, or actuator control. This distributed design not only allows independent development and testing of each module but also improves fault tolerance, maintainability, and scalability. The following sections explore the motivations behind using ROS2, its key features, and comparisons with previous versions.

4.4.1 Why ROS?

Conventional approaches to robotic system design often involve the development of tightly coupled software architectures, wherein navigation, sensor integration, actuator control, and user interfaces are implemented within a single monolithic codebase. While such methods may suffice for simple applications, they present significant limitations in terms of scalability, maintainability, and modularity. Introducing new hardware components—such as LiDAR sensors, additional actuators, or vision systems—typically necessitates extensive code modifications, often resulting in fragile systems prone to synchronization issues, communication failures, and poor reusability.

In contrast, the Robot Operating System (ROS), and particularly its modern iteration ROS2, provides a modular and distributed middleware framework specifically designed to address these challenges. ROS2 supports asynchronous, peer-to-peer communication between loosely coupled nodes, each responsible for a specific task. This architectural paradigm facilitates the integration of diverse hardware and software components while maintaining clear separation of concerns and ensuring system robustness.

For the present project, ROS2 functions as the central coordination layer, orchestrating interactions among the robot's subsystems—including navigation algorithms, the RPLiDAR sensor, the Raspberry Pi 4 onboard computer, and the Arduino Mega microcontroller. The use of standardized message-passing interfaces (topics, services, and

actions) simplifies the development process and significantly enhances system clarity and debuggability.

Furthermore, ROS2 provides built-in support for real-time capabilities and deterministic behavior, which are essential for tasks such as obstacle avoidance, path planning, and localization. Development and testing are further supported by simulation and visualization tools such as Gazebo and RViz2, which allow for comprehensive validation of robot behavior in both virtual and real-world environments.

The adoption of ROS2 also ensures long-term extensibility of the system. The modular nature of ROS nodes allows for the seamless addition of new features—such as depth cameras, UV-C sterilization modules, or semantic perception systems—without requiring fundamental architectural changes. This capability is particularly valuable for projects intended to evolve over time or adapt to varying deployment scenarios.

In summary, ROS2 offers a robust, scalable, and future-oriented framework that addresses the inherent limitations of traditional robotic software development methods. Its adoption in this project provides a solid foundation for building a reliable, flexible, and extensible disinfection robot suited for real-world indoor environments.

4.4.2 What is ROS?

The Robot Operating System (ROS) is not a traditional operating system but a flexible and powerful middleware designed to support the development of complex robotics applications. Originally developed by Willow Garage and maintained today by Open Robotics, ROS provides a standardized interface for robotic hardware and software components. It facilitates modularity and scalability by organizing robotic software into small, loosely coupled units called nodes, which communicate through topics, services, and actions.

At its core, ROS includes libraries for message passing, hardware abstraction, sensor integration, package management, and simulation tools. It has become the standard in robotics research and development due to its active community, strong ecosystem, and support for numerous robots and sensors. ROS simplifies the otherwise complex task of integrating various hardware components — motors, sensors, cameras, and microcontrollers — and managing their communication efficiently.

4.4.3 ROS2 System Architecture: Nodes, Topics, and Transforms

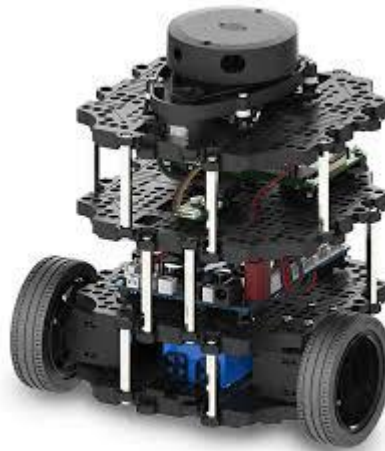


Figure 2.2: *Example of a mobile robot platform used to demonstrate ROS2 architecture with mounted LiDAR sensor and modular layer structure for perception, planning, and control.*

The ROS2 system architecture implemented in this robot adheres to a distributed node-based design, where each node is responsible for a distinct task within the robot's perception, planning, or control subsystems. This modularity not only supports system scalability and maintainability but also ensures fault tolerance by isolating functionality into independently managed processes.

Central to the perception pipeline is the SLAM node, which performs real-time mapping and localization using data acquired from the RPLiDAR A1 sensor. This sensor continuously publishes laser scan data to the `/scan` topic, which the SLAM Toolbox node subscribes to. By processing this input, the node incrementally constructs a 2D occupancy grid and publishes it on the `/map` topic. This map is then utilized by the localization subsystem, typically configured with the AMCL (Adaptive Monte Carlo Localization) node or its lifecycle-managed equivalent, which estimates the robot's pose relative to the static map using particle filtering techniques.

The navigation subsystem is orchestrated through the Nav2 stack, comprising the `planner_server` and `controller_server`. These nodes ingest live pose estimates, map data, costmaps, and sensor observations to compute motion plans and obstacle-free trajectories. The global planner determines the optimal path to a given goal, while the local planner

dynamically adjusts this path in response to real-time environmental changes. The resulting velocity commands are published to the `/cmd_vel` topic. These messages are intercepted by a dedicated serial interface node, which forwards them to the Arduino Mega. The Arduino interprets the velocity commands and actuates the motors via the L298N driver, ensuring accurate and responsive robot movement.

A fundamental component of ROS2 communication is the transform (TF) system, which manages the coordinate frames between various parts of the robot. The transform tree for this robot follows the conventional structure of `/map` \rightarrow `/odom` \rightarrow `/base_link`, with additional child frames for `/base_laser` and `/camera_link`. This hierarchical frame model ensures that sensor readings, motion commands, and localization data are all referenced consistently in space. The TF broadcaster nodes publish these transformations in real time, enabling downstream components such as RViz2 and the costmap layers to correctly interpret spatial relationships and maintain coherence during navigation and perception tasks.

In addition to autonomous navigation, a Bluetooth override mechanism is integrated into the architecture through a dedicated `bluetooth_listener_node`, which listens for manual control commands and relays them to the Arduino. This allows seamless transitions between manual and autonomous operation without restarting any nodes or reinitializing the system state.

Overall, the robot's ROS2 architecture enables tightly synchronized data flow between perception, planning, and actuation modules. The combination of topic-based messaging, structured TF frames, and modular node composition makes the system highly adaptable to new hardware, upgraded algorithms, or extended functionalities, without necessitating a complete architectural redesign.

4.4.4 ROS2 Control Architecture: Functional Layer Breakdown

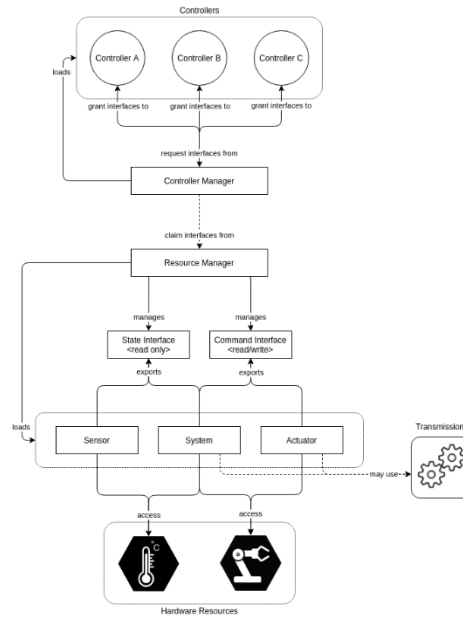


Figure 2.3: : ROS2 Control Architecture showing the layered interaction between controllers, interfaces, and hardware components.

In **Figure 2.3**. The ROS 2 Control Framework forms the backbone of the robot’s low-level actuation and hardware communication system. It is engineered around a modular, real-time compatible architecture that divides responsibilities across well-defined functional layers. These include the Controllers Layer, the Management and Interface Layer (comprising the Controller Manager and Resource Manager), and the Hardware Abstraction Layer. This layered approach ensures separation of concerns, making the system highly maintainable, extensible, and suitable for deployment in time-sensitive robotic applications.

At the topmost level resides the Controllers Layer, which hosts one or more active control nodes—each responsible for managing specific subsystems or behaviors. These controllers are implemented as ROS 2 nodes or plugins and execute real-time strategies such as position, velocity, or trajectory control. Although they implement control logic, controllers are intentionally decoupled from the underlying hardware. Instead, they interact through abstracted command and state interfaces provided by lower layers, which

allows the same control logic to be reused across different robots as long as compatible interfaces are exposed.

The Controller Manager is the orchestration layer that oversees the entire lifecycle of controllers. It is responsible for loading, configuring, starting, stopping, and unloading controllers at runtime. Upon initialization, each controller registers its hardware interface requirements—such as joint position or velocity interfaces—via the Controller Manager. To ensure system integrity, the Controller Manager communicates with the Resource Manager to check for conflicting access requests. It guarantees that multiple controllers do not simultaneously command the same hardware resources unless explicit coordination is in place.

Beneath the Controller Manager lies the Resource Manager, which acts as the intermediary between software controllers and physical hardware. When a controller requests access to an interface, the Resource Manager validates and allocates the corresponding command or state interface. Two principal types of interfaces are defined: State Interfaces, which are read-only and allow controllers to monitor hardware conditions such as joint angles, temperatures, or sensor states; and Command Interfaces, which are read-write and enable controllers to issue actuation commands to motors, servos, or other actuators. This structured abstraction allows the robot's control logic to remain insulated from hardware-specific details.

The hardware abstraction layer is composed of three core components: Sensor, System, and Actuator modules. Sensor modules represent read-only devices, such as temperature sensors, inertial measurement units, or laser rangefinders, and only expose state interfaces for observation. System modules are hybrid abstractions that encapsulate both sensing and actuation logic—commonly used for mobile robot bases or robotic arms—providing a unified interface to multiple types of hardware resources. Actuator modules represent devices capable of mechanical movement and generally expose command interfaces, and optionally, state interfaces for feedback-driven control. Each hardware unit is implemented as a plugin conforming to the ROS 2 hardware interface specifications, allowing seamless swapping or extension of hardware with minimal system reconfiguration.

Underpinning these components is the hardware resource layer, which interfaces directly with physical devices such as motors, servos, and sensors. For robots with mechanical complexity, such as differential drive systems or articulated arms with gearboxes, a transmission layer can be optionally introduced. This layer translates control inputs—such as rotational velocity—from actuator-level commands into meaningful joint-level behaviors. The transmission model accounts for mechanical linkages and ensures accurate mapping between hardware commands and robot kinematics, which is essential for precise control in real-world scenarios.

The data and control flow within the ROS 2 Control architecture is deterministic and hierarchically regulated. A controller first requests access to one or more interfaces via the Controller Manager. The Manager then coordinates with the Resource Manager, which verifies and grants access to the appropriate interfaces linked to hardware components. These interfaces serve as the conduit between high-level logic and the actual hardware, enabling real-time control loops to operate within isolated and well-scoped boundaries. This flow ensures that only authorized controllers interact with hardware and that every interface exchange adheres to ROS 2 communication standards.

One of the most powerful aspects of this design is its modularity. Each layer—controller, manager, and hardware abstraction—can be developed, tested, and maintained independently. This facilitates code reuse across projects and simplifies integration with simulation environments such as Gazebo or Ignition. Additionally, the architecture supports real-time processing by enabling low-latency threads or processes for time-critical operations like PID control. These features make the ROS 2 Control Framework exceptionally suited for robots that demand reliability, flexibility, and deterministic behavior in both research and production environments.

4.4.5 Writing Launch Files and Startup Scripts

To efficiently manage the various ROS2 nodes and their configurations, the robot system utilizes Python-based launch files. These launch files define which components of the software stack should be executed, along with their corresponding parameters, topic remappings, and namespace declarations. A central launch file named `disinfection_robot.launch.py` serves as the entry point for initializing the core functionalities of the robot. It is responsible for launching essential nodes including the

SLAM Toolbox (for mapping mode) or the AMCL node (for localization mode), the Nav2 lifecycle manager, the RPLiDAR driver, static transform publishers, and custom interfaces such as the Bluetooth listener node that interacts with the manual control system.

One of the key strengths of ROS2 launch files is their ability to support conditional logic, enabling the same launch file to be used flexibly for different operational modes. For instance, developers can specify through command-line arguments whether the robot should boot into SLAM mapping mode or localization mode using a pre-saved map. This flexibility reduces code duplication and simplifies deployment across different use cases.

Beyond launch files, the system employs Bash-based startup scripts to automate service initialization during system boot. These scripts perform several critical tasks: they set the required ROS2 environment variables, invoke ros2 launch commands to start the main navigation stack, initialize and verify the availability of USB-connected devices such as the RPLiDAR sensor, and open serial communication ports to interface with the Arduino. Some scripts are also configured to start system monitors or loggers to track robot behavior or failures.

To enable full autonomy from the moment the robot powers on, these scripts are either placed in `/etc/rc.local` or registered as systemd services within the Raspberry Pi OS environment. This ensures that the complete ROS2-based software stack is brought online automatically, without any user interaction. Such automated startup behavior is essential for real-world applications in hospitals, offices, or commercial facilities where the robot must begin disinfection routines immediately after being powered on or rebooted. The combination of dynamic ROS2 launch files and robust Bash startup scripts contributes to the reliability and readiness of the robot during live deployments.

4.4.6 Comparison Between ROS1 and ROS2

The Robot Operating System (ROS) has undergone significant evolution since its inception. ROS1, introduced over a decade ago, laid the foundation for modern robotic software development by introducing modularity, reusable components, and a standardized communication model based on message passing. It quickly gained traction in academia and research institutions due to its ease of use, extensive community support,

and compatibility with a wide array of robotic platforms. However, despite its widespread adoption, ROS1 was primarily designed as a research-oriented tool, with several architectural and performance limitations that restrict its suitability for production-grade, real-time applications.

One of the key limitations of ROS1 lies in its communication architecture. ROS1 employs a centralized master node responsible for managing the registration and coordination of all other nodes within the system. This design introduces a single point of failure and can become a bottleneck in systems with large numbers of nodes or high-frequency message exchange. Additionally, ROS1's reliance on TCP-based communication and lack of real-time guarantees further limit its applicability in mission-critical or time-sensitive robotic deployments.

In contrast, ROS2 was developed from the ground up to address these limitations and enable more robust, scalable, and production-ready robotic systems. The most fundamental change in ROS2 is the adoption of the Data Distribution Service (DDS) as its core communication protocol. DDS is a middleware standard designed for distributed, real-time systems, enabling peer-to-peer, decentralized communication without the need for a master node. This shift significantly improves system resilience, reduces latency, and allows better scalability across multiple robots and networked environments.

Moreover, ROS2 introduces native support for real-time performance, a critical requirement for tasks such as control loops, obstacle avoidance, and sensor feedback processing. It achieves this by supporting multi-threaded execution, deterministic callback scheduling, and compatibility with real-time operating systems (RTOS). These features provide developers with fine-grained control over execution behavior and ensure consistent response times in dynamic and unpredictable environments.

Another key improvement in ROS2 is cross-platform compatibility. Unlike ROS1, which was limited to Linux-based systems, ROS2 supports Linux, Windows, and macOS, broadening its usability and facilitating integration with diverse development ecosystems. ROS2 also includes enhanced security mechanisms—such as encrypted communication, authentication, and access control—making it suitable for deployment in commercial and industrial settings where data integrity and protection are paramount.

The architectural redesign of ROS2 extends beyond performance and security enhancements. It provides a more modern and flexible launch system (based on Python), improved parameter management, better namespace isolation, and support for lifecycle nodes, which allow for more predictable state transitions in complex robotic systems. These improvements contribute to higher code reusability, easier debugging, and smoother system integration.

Given these substantial advantages, ROS2 was selected as the middleware for the disinfection robot described in this project. Its ability to support hybrid control architectures, interface with multiple hardware components, and operate in both simulation and real environments with minimal reconfiguration makes it an ideal choice for autonomous service robots operating in dynamic indoor environments.

4.4.7 ROS2 Humble Hawksbill and Ubuntu Compatibility



Figure 2.4: ROS2 Humble Hawksbill – the chosen LTS distribution for this project, ensuring stability and compatibility with Ubuntu 22.04.

For this project, the selected ROS2 distribution is Humble Hawksbill, a Long-Term Support (LTS) version officially released in May 2022 and maintained until May 2027. Choosing an LTS distribution ensures long-term stability, security updates, and compatibility with widely used development tools and packages throughout the system's life cycle. This is especially important for academic and industrial research projects that require consistent performance over extended periods.

Humble Hawksbill is designed to operate optimally on Ubuntu 22.04 (Jammy Jellyfish), a stable and widely adopted Linux distribution. This pairing offers developers a robust and up-to-date software environment with access to tested ROS packages, improved kernel support, and better hardware compatibility. Furthermore, Humble provides seamless integration with tools like Gazebo Fortress for simulation and the Nav2 Stack for autonomous navigation, both of which are critical for prototyping and validating robotic behavior.

Several notable enhancements distinguish Humble from its predecessors. The improved node composition model allows multiple ROS2 nodes to be loaded dynamically into a single process, reducing inter-process communication latency and memory consumption. This is particularly useful when deploying the robot on a resource-constrained edge computing platform like the Raspberry Pi 4, where efficiency is paramount. By composing nodes such as the controller server, planner server, and recovery behaviors into one executable, system initialization becomes faster and runtime performance more predictable.

The enhanced parameter handling system in Humble supports live updates to critical configuration variables. For instance, the operator may modify the maximum velocity, inflation radius, or goal tolerances of the navigation stack without requiring a system reboot. This capability enables adaptive behavior tuning during field deployment, especially in dynamic environments like hospitals or offices with frequent layout changes.

Another improvement is the modern Python-based launch system, which simplifies the process of orchestrating complex multi-node applications. Using conditional logic and event handlers, the robot can be launched in either mapping or localization mode with minimal changes to the launch file. This flexibility is essential when switching between development, testing, and operational workflows.

Humble also brings a wider scope of compatibility for hardware peripherals, including updated drivers and support for newer LIDARs, IMUs, and depth cameras. This facilitates future integration of additional sensors into the disinfection robot platform, such as 3D SLAM modules, environmental gas sensors, or QR-code-based localization beacons, without requiring custom device drivers or ROS wrappers.

From a stability standpoint, ROS2 Humble has significantly matured the Nav2 stack and SLAM Toolbox. The navigation system has improved failover behaviors, better recovery plugin support, and enhanced local planner algorithms. Likewise, SLAM Toolbox now features better loop closure detection, map serialization tools, and online/offline mode switching, all of which are critical for deploying the robot in multi-room indoor facilities.

Moreover, Humble supports real-time execution via the ROS 2 Real-Time Working Group initiatives. While not fully deterministic on general-purpose operating systems like Ubuntu, the framework allows developers to prioritize specific threads or nodes using executor models that align with real-time requirements. This feature can be leveraged in future upgrades where deterministic control of motion or sensing is needed.

In conclusion, ROS2 Humble Hawksbill offers an optimal combination of long-term support, ecosystem maturity, and critical features required for deploying intelligent autonomous robots. In the context of this disinfection and scent-diffusing robot, Humble enables seamless switching between manual and autonomous control modes, robust mapping and navigation, real-time adaptability, and future extensibility—all of which are vital to ensure safe and consistent operation in real-world indoor environments.

4.4.8 Raspberry Pi OS and Network Configuration



Figure 2.5: *Raspberry Pi setup used for running the robot's operating system and managing network communication through USB and Ethernet connections.*

The disinfection robot utilizes a Raspberry Pi 4 as its onboard processing unit, running Raspberry Pi OS (64-bit), a Debian-based Linux distribution tailored for ARM architectures. This operating system offers a favorable balance between performance, stability, and compatibility, making it a suitable platform for deploying ROS2-based applications. The Raspberry Pi OS environment is configured with essential services such as SSH and VNC, which enable secure remote access for development, diagnostics, and field monitoring. Additionally, custom udev rules are defined to ensure consistent USB port assignment for critical peripherals, such as the RPLiDAR sensor and Arduino Mega, regardless of the order in which devices are connected or initialized at boot. The system also supports Python 3.10 and above, ensuring compatibility with ROS2 launch scripts and Python-based nodes that rely on modern language features. In performance-sensitive deployments, optional real-time kernel patches can be applied to enhance task scheduling priority for latency-critical threads.

Network communication is a vital component of the system, particularly in multi-node or remote-monitoring scenarios. The Raspberry Pi supports both wired Ethernet and wireless Wi-Fi connections, and the configuration is designed to utilize either depending on the operational context. In stable, fixed environments—such as laboratories or clinical settings—Ethernet is preferred due to its superior reliability, lower latency, and higher bandwidth. Conversely, during mobile demonstrations, ad-hoc testing, or in environments without wired infrastructure, the system defaults to Wi-Fi connectivity. To maintain node visibility in ROS2's distributed architecture, static IP addresses are assigned when operating over wireless networks, ensuring consistent node discovery and reducing reliance on dynamic network resolution protocols.

The robot's ROS2 communication is built on the Fast DDS (formerly Fast RTPS) middleware, which supports real-time peer-to-peer discovery, Quality of Service (QoS) configuration, and efficient topic broadcasting across dynamic IP environments. This middleware layer ensures reliable inter-node communication, even under varying network conditions. During development and deployment, a suite of diagnostic tools is used to verify network health and communication efficiency. Utilities such as ping are used to measure basic connectivity and latency, while ROS-native tools like `ros2 topic hz`, `ros2 topic echo`, and `rqt_graph` provide detailed insight into message frequencies, publishing node health, and topic interconnectivity. This diagnostic infrastructure allows

for early detection of communication bottlenecks, misconfigured topics, or lost packets, enabling developers to optimize system performance and reliability across different network topologies.

4.4.9 ROS2 Node Graph and Data Flow

The internal software architecture of the disinfection robot is structured as a modular graph of ROS2 nodes, each assigned a specific functional responsibility within the autonomous navigation and control pipeline. These nodes communicate through named topics using the ROS2 publish-subscribe model, forming a directed acyclic data flow that enables concurrent and asynchronous operation across the system.

At the sensory input layer, the `rplidar_node` is responsible for interfacing with the RPLiDAR A1 sensor and publishing laser scan data to the `/scan` topic in the `sensor_msgs/LaserScan` format. This scan data is consumed by either the `slam_toolbox` node—when in mapping mode—or the `amcl` node when operating in localization mode. Both of these nodes subscribe to `/scan` and `/tf` to perform pose estimation and mapping. The SLAM Toolbox, when active, publishes the evolving occupancy grid map to `/map` and the robot’s estimated pose to the relevant TF frames. In contrast, AMCL focuses on computing the robot’s pose within a static map, publishing pose information for downstream navigation logic.

For path planning and goal execution, the `nav2_bt_navigator` node manages high-level decision-making through a behavior tree architecture. It listens for goal inputs, either from the user interface or command-line tools, and coordinates subsequent planning and control behavior. The `planner_server` node takes these goals and generates a global path by analyzing the `/map` and `costmap` layers, publishing the output to the `/plan` topic. Once a valid path is available, the `controller_server` node processes it and computes velocity commands in real time, which are then published to the `/cmd_vel` topic. These velocity commands represent linear and angular movement instructions and are consumed by the low-level motion controller.

The `robot_state_publisher` node ensures the continuous broadcast of the robot’s transform tree (`/tf`), which defines the spatial relationships between all reference frames—such as

/base_link, /odom, /map, and any sensor-attached frames. This transform data is critical for maintaining spatial consistency across sensor inputs, pose estimates, and actuator outputs.

On the actuation side, a custom Arduino interface node subscribes to /cmd_vel and relays these velocity commands to the Arduino Mega over a serial connection. The Arduino then interprets and applies them to the motor driver circuit, which controls the robot's movement. Parallel to this, a Bluetooth listener node operates independently, subscribing to a custom /bluetooth_cmd topic. This node processes manual override commands, which may include motion instructions or relay toggles for the disinfection and perfuming systems. These commands are routed either directly to the Arduino or passed into a relay control module, depending on the nature of the command.

The complete node graph is defined and managed via a central ROS2 launch file. Each node's parameters, topic remappings, and lifecycle configurations are declared explicitly, enabling consistent and predictable startup. Visualization and monitoring tools like rqt_graph and RViz2 allow real-time inspection of the active node network and data flows. This modular and loosely coupled architecture simplifies system-level debugging and supports plug-and-play extensibility—for example, swapping out the global planner or updating localization techniques without modifying motor control or hardware interface logic.

4.4.10 System Execution Flow or Logic Design

The internal logic architecture of the disinfection robot is designed around a deterministic control flow that supports both autonomous navigation and manual override operation in a seamless and coordinated manner. Upon system startup, the execution flow begins with a comprehensive hardware initialization phase. In this stage, all critical components—including the RPLiDAR sensor, DC motor drivers, relay modules for humidifier control, the Bluetooth HC-05 interface, and communication buses—are activated and tested for readiness. Simultaneously, the ROS2 launch infrastructure initiates the required nodes and parameters, setting the foundation for either mapping or localization depending on the deployment mode.

Following initialization, the robot transitions into a standby state where it passively awaits high-level instructions from the user or system triggers. If the robot is operating in autonomous mode, the ROS2 Nav2 stack is activated. In this mode, the robot loads a prebuilt static map if working in a known environment or initiates real-time map generation via the SLAM Toolbox in exploration scenarios. Localization is achieved using AMCL or SLAM-published pose estimates, which are consumed by the global planner to compute an optimal path to the user-defined goal. The robot then executes this path using a local planner, such as DWB, which generates real-time velocity commands based on current obstacle data, trajectory optimization, and LiDAR-based collision detection. These commands are published to the `/cmd_vel` topic, which is received by the Arduino-based motion control system for execution.

In contrast, when the system is switched to manual mode—either through a GUI toggle, a mobile application, or a startup script parameter—the robot suppresses all autonomous behavior and suspends the Nav2 planner. Instead, it opens a communication channel with the Bluetooth module to listen for ASCII-based control commands issued by the operator. These commands are interpreted by the Arduino to control motor movement directly, and to toggle the two relay-controlled humidification units independently. Manual mode provides real-time feedback to the operator and is particularly useful for debugging, precise spot sterilization, or operating in environments not yet mapped or considered unsuitable for autonomous navigation.

At any point in the execution, the user may switch between autonomous and manual control modes. This transition is handled gracefully through a mode management protocol embedded within both the Arduino firmware and ROS2 startup logic. Safety is preserved through motion interlocks, timeout-based watchdogs, and operational prioritization, ensuring that control handover does not lead to undefined or hazardous behavior.

This hybrid control flow—characterized by a bifurcated decision tree for manual and autonomous operation—provides a flexible yet robust logic backbone for the robot's disinfection duties. It allows for reliable performance in varied operational contexts, supports continuous testing and development cycles, and minimizes system risk by clearly segmenting responsibilities between subsystems and control layers.

4.4.11 Manual Control via Bluetooth and HC-05

Although the robot is designed to operate autonomously, certain real-world scenarios necessitate manual override capability. These include tasks such as performing spot disinfection in hard-to-reach areas, guiding the robot in environments not previously mapped, or conducting supervised testing during development. To accommodate such needs, a manual control mode is implemented using a Bluetooth communication interface, allowing a human operator to control the robot directly via a mobile device.

The manual control system is powered by an HC-05 Bluetooth module connected to the Arduino Mega, which acts as the low-level controller for both motion and peripheral actuation. The module operates using a standard serial interface (UART) and communicates over Bluetooth Classic (SPP profile), providing a simple and reliable link between the robot and any compatible smartphone application. In this project, the Bluetooth Electronics app is used due to its customizable interface and compatibility with ASCII-based command systems.

On the software side, the Arduino listens continuously for incoming Bluetooth commands via its UART interface. Commands are encoded as simple single-character or multi-character strings—for example, 'F' for forward motion, 'B' for backward, 'L' and 'R' for turning, and 'S' for stopping all motion. This command set is parsed in real time and converted into appropriate digital outputs to control the H-bridge motor drivers connected to the drive wheels. The system also includes command debounce logic and mutual exclusion routines to avoid conflicting motor instructions that could otherwise damage hardware or lead to erratic behavior.

In addition to movement, the robot's two humidification units—used for chemical disinfection and perfuming—are also controllable via Bluetooth. Each humidifier is connected to a relay module, and the Arduino can toggle these relays upon receiving specific commands such as 'H1' or 'H2'. These actuators are isolated from the main logic through opto-isolated relays to prevent voltage spikes from damaging the microcontroller. Each humidifier is assigned a distinct role in the system, and their activation can be managed independently of movement, enabling precise, localized treatment in specific areas.

To ensure smooth switching between autonomous and manual modes, a mode-handling protocol is embedded in the Arduino firmware. When Bluetooth commands are received, the system temporarily halts any navigation commands from the Raspberry Pi and grants control priority to the manual interface. This switching is designed to be reversible, allowing seamless transition back to autonomous mode once manual operation is no longer needed. Safety checks are also implemented to prevent mode switching during high-speed movement or critical operations like loop closure or SLAM optimization.

This hybrid control architecture—blending autonomous ROS2-based navigation with real-time Bluetooth override—provides a significant degree of operational flexibility. It empowers users to intervene when necessary without compromising the autonomy pipeline. Additionally, it allows for field-specific disinfection strategies to be implemented without redeploying or reprogramming the robot, making the system adaptable to diverse use cases, including hospitals, offices, or event venues.

4.4.12 Bluetooth Command Protocol

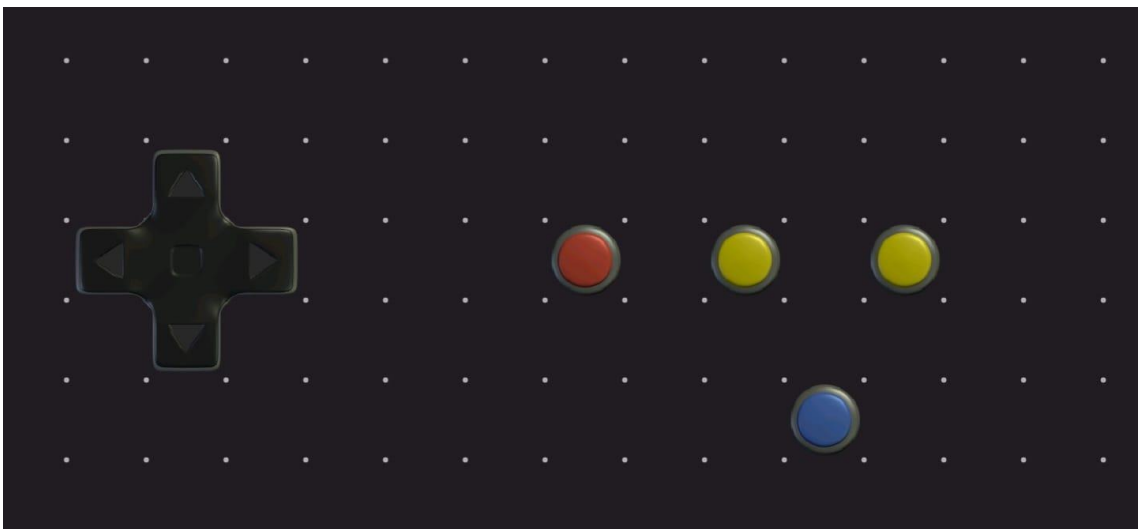


Figure 2.6: *Bluetooth control interface used to send serial commands from the mobile device to the robot via the HC-05 module for manual operation.*

To enable manual control of the robot, a simple and robust Bluetooth command protocol is implemented over serial communication between the mobile device and the HC-05 module connected to the Arduino. This protocol is designed to be human-readable, lightweight, and easy to extend for future functionality. It allows for intuitive control by

sending predefined ASCII command strings from a smartphone application to the robot's onboard controller.

The protocol supports a set of high-level commands, each corresponding to a specific action. These include motion directives such as FWD for moving forward, BACK for reverse motion, LEFT and RIGHT for turning, and STOP to halt all motion. In addition to mobility, the system includes commands for peripheral control: H1_ON and H1_OFF are used to activate and deactivate the disinfection humidifier, while H2_ON and H2_OFF control the perfume humidifier.

Each command is transmitted as a plain-text string terminated by a newline character (\n). Upon receipt, the Arduino parses the incoming command using string-matching logic and immediately invokes the appropriate control routine. This architecture ensures low latency and reliable execution, even in dynamic or time-sensitive operational contexts.

Optionally, the system can send acknowledgment messages (e.g., ACK: H1_ON Executed) back to the mobile device. These feedback messages confirm successful command execution and can be displayed in the mobile app through terminal widgets, enhancing usability and operator confidence.

The simplicity of this protocol makes it accessible to non-technical users who may lack prior experience with robotics or embedded systems. By leveraging familiar mobile interfaces and intuitive command syntax, the robot becomes operable in a wide range of scenarios without requiring extensive user training or configuration.

4.5 AI INTEGRATION AND OBJECT DETECTION MODEL

To enhance the robot's perception capabilities, an AI-powered object detection module was integrated using the YOLOv8 (You Only Look Once version 8) architecture. This deep learning model allows the robot to detect high-contact surfaces—such as chairs, tables, door handles, and light switches—with high precision and real-time efficiency.

4.5.1 Purpose and Benefits of AI Integration

The integration of AI serves multiple objectives:

1. **Precision in Targeting:** AI enables accurate identification of critical surfaces, reducing unnecessary disinfectant usage.
2. **Research-Based Prioritization:** The model is trained on surfaces proven to carry high pathogen loads.
3. **Efficiency:** Intelligent prioritization reduces operation time and increases disinfection coverage.
4. **Environmental Adaptation:** The robot adjusts its disinfection path in real-time as the environment changes.
5. **Consistency:** AI eliminates human error, ensuring uniform performance across all sessions.
6. **Cost-Effectiveness:** By minimizing chemical waste and optimizing usage, operational costs are significantly reduced.
7. **Safety Enhancement:** AI detects surfaces that may be overlooked or are difficult to access manually.
8. **Autonomous Functionality:** Enables near fully autonomous disinfection with minimal human intervention.

4.5.2 Model Architecture

YOLOv8 was selected due to its balance of speed, accuracy, and suitability for real-time robotic deployment. Key features include:

- High inference speed for real-time detection
- Lightweight design for embedded platforms
- Proven performance in mobile and embedded robotics

4.5.3 Dataset Development

A dataset of **9,996** images was constructed, combining custom images from indoor environments and pre-labeled public datasets from Roboflow. The four target classes were:

- **Chairs**
- **Tables**
- **Door Handles**
- **Light Switches**

Each class had approximately 1005 labeled images. The dataset was split into:

- **Training:** 8,964 images
- **Validation:** 684 images
- **Testing:** 348 images

Augmentations included flips, rotations, shearing, brightness/saturation adjustments to enhance generalization

4.5.4 Training Configuration

The YOLOv8 small (YOLOv8s) model was trained with the following hyperparameters:

- **Epochs:** 50
- **Batch Size:** 32
- **Learning Rate:** 0.001
- **Optimizer:** Adam
- **Image Size:** 500×500 pixels
- **Training Environment:** Roboflow cloud with GPU acceleration

Model performance was monitored using metrics such as precision, recall, and mAP.

4.6 MODEL IMPLEMENTATION AND CODE WORKFLOW

To implement the object detection model for identifying high-touch surfaces, a structured coding pipeline was developed using the YOLOv8 framework. The process covers environment setup, dataset integration, model training, and real-time inference.

4.6.1 Environment Setup and Dataset Handling

```
[1] !nvidia-smi

Tue Jun 3 15:54:06 2025

+-----+ Driver Version: 550.54.15  CUDA Version: 12.4  +-----+
| NVIDIA-SMI 550.54.15 |
+-----+-----+-----+-----+-----+-----+
| GPU  Name      Persistence-M | Bus-Id  Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |      Memory-Usage | GPU-Util  Compute M. |
|=====+=====+=====+=====+=====+=====+=====+
|  0  Tesla T4           Off | 00000000:00:04:0 Off |   0%  Default  MIG M. |
| N/A   48C    P8          10W / 70W |  0MiB / 15360MiB |   0%          N/A  |
+-----+-----+-----+-----+-----+-----+

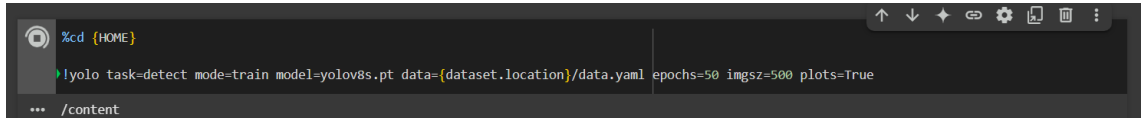
Processes:
+-----+-----+-----+-----+-----+-----+-----+
| GPU  GI  CI       PID  Type  Process name                        GPU Memory |
| ID   ID  ID                                   name                       Usage      |
+-----+-----+-----+-----+-----+-----+-----+
| No running processes found |
+-----+-----+-----+-----+-----+-----+-----+

```

Figure 2.7: GPU environment setup using NVIDIA Tesla T4, confirming CUDA compatibility for training YOLOv8 models with the annotated dataset.

installed the **Ultralytics YOLOv8** library (v8.2.103) and **Roboflow** (v1.1.48). Dataset was downloaded via the Roboflow API with predefined train, validation, and test splits. Dataset included **9,996 images** annotated in YOLO format, covering four object classes (Chairs, Tables, Door Handles, and Light Switches).

4.6.2 Model Training Configuration

A terminal window with a dark background. The prompt is '%cd {HOME}'. The command entered is 'yolo task=detect mode=train model=yolov8s.pt data={dataset.location}/data.yaml epochs=500 imgsz=500 plots=True'. The cursor is at the end of the command. The bottom of the terminal shows '*** /content'.

```
%cd {HOME}
yolo task=detect mode=train model=yolov8s.pt data={dataset.location}/data.yaml epochs=500 imgsz=500 plots=True
*** /content
```

Figure 2.8: YOLOv8s training command configuration using a custom dataset with 500 epochs, 500×500 image size, and enabled augmentation and plotting features.

- Model: YOLOv8s (small), pretrained.
- Epochs: 500 (with early stopping).
- Image Size: 500×500.
- Augmentations: Enabled (flips, rotations, brightness/saturation).
- Plots: Generated performance metrics automatically

Training progress was visualized using built-in tools showing:

- Loss Curves: Classification, box, and distribution focal losses.
- Precision & Recall: Training and validation curves.
- mAP Scores: mAP@0.5 and mAP@0.5:0.95 over time.

4.6.3 Output and Evaluation Artifacts

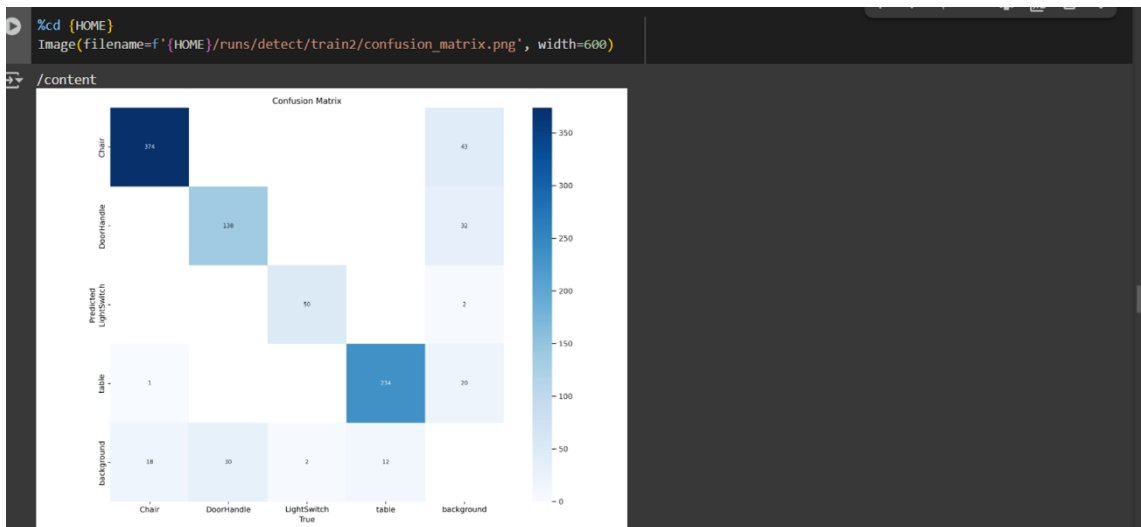


Figure 2.9: Confusion matrix visualization showing the model's classification performance across object classes after training.

Upon completion, the following outputs were generated. Figure 2.9 and Figure 3.1 illustrates the training performance:

- **best.pt**: Best-performing model weights
- **results**: Graphical plots of training performance
- **confusion_matrix**: Visualization of class prediction performance
- **predict**: Folder containing test images with drawn bounding boxes

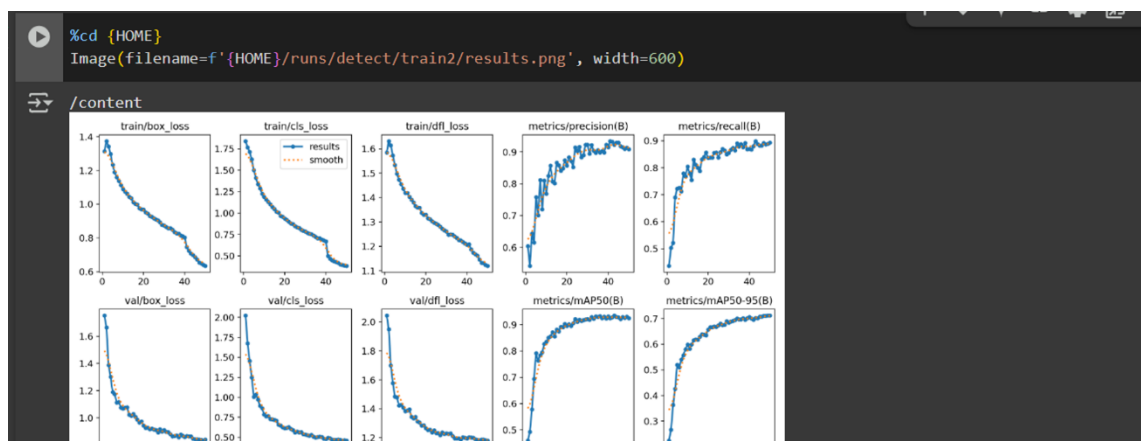


Figure 3.1: Training performance metrics plotted during YOLOv8 model training, including box loss, classification loss, DFL loss, precision, recall, and mean average precision (mAP) scores.

4.6.4 Testing and Deployment Code

- Ran inference on test set using the saved best weights
- Set confidence threshold to **0.25**
- Results saved with bounding boxes for validation

This code structure ensured reproducibility, easy debugging, and robust integration with the robot's real-time perception system.

4.7 KINECT-BASED RGB INPUT INTEGRATION

4.7.1 System Requirements:

1. Hardware:

- **Microsoft Kinect v1 (Xbox 360 version).** USB 2.0+ port (recommended powered USB hub). 12V 1.08A power adapter (if using original Kinect power cable)
- **Raspberry Pi 4/5 (Recommended).** Minimum 4GB RAM. 32GB+ SD card (UHS-I Class 10). Active cooling recommended

2. Software:

- **OS:** Raspberry Pi OS (64-bit) Bullseye
- **Python:** 3.9-3.11

Driver Installation:

IN Figure 4, Figure 5, Figure 6: To enable communication between the Microsoft Kinect v1 (Xbox 360 version) and a Linux-based system such as the Raspberry Pi, a series of software dependencies and USB permissions must be correctly configured. The following steps detail the installation of the libfreenect open-source driver and the required USB access rules to ensure compatibility with Python and computer vision frameworks such as OpenCV and YOLOv8.

The installation procedure begins with the installation of development libraries including libusb, python3-dev, and associated build tools. Once the environment is prepared, the

libfreenect driver is cloned from its official repository and compiled with Python 3 support enabled using the `-DBUILD_PYTHON3=ON` flag.

To ensure that all users have proper access to the Kinect's motor, camera, and audio interfaces, appropriate udev rules are defined in a custom rules file (`51-kinect.rules`). These rules grant the necessary read/write permissions for USB devices identified by Kinect's vendor ID `045e`.

After defining the rules, they are activated by reloading the udev system and triggering the rules.

```
bash
# Install core dependencies
sudo apt update
sudo apt install -y \
    cmake \
    libusb-1.0-0-dev \
    freeglut3-dev \
    libxmu-dev \
    libxi-dev \
    python3-dev \
    cython3

# Install libfreenect
git clone https://github.com/OpenKinect/libfreenect.git
cd libfreenect
mkdir build
cd build
cmake .. -DBUILD_PYTHON3=ON
make
sudo make install
sudo ldconfig

# Set USB permissions
sudo nano /etc/udev/rules.d/51-kinect.rules
```

Figure 5: Installation and configuration steps for libfreenect and udev rules to enable Kinect device access on Linux systems.

```
text
# Kinect motor
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02b0", MODE="0666"
# Kinect audio
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02ad", MODE="0666"
# Kinect camera
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02ae", MODE="0666"
```

Figure 5: Custom udev rules assigning read/write permissions for Kinect motor, audio, and camera USB interfaces using vendor ID `045e` and respective product IDs.

```
bash
sudo udevadm control --reload-rules
sudo udevadm trigger
```

Figure 6: Commands used to reload and apply udev rules, ensuring immediate activation of new USB device permissions.

4.7.2 Python Environment Configuration for Kinect-YOLOv8

Integration:

Figure 7: To ensure seamless compatibility between the Microsoft Kinect data stream and the YOLOv8 object detection model, a dedicated Python environment was configured. This environment was established using a virtual environment named `kinect_yolo`, which isolates dependencies and prevents version conflicts.

Specific package versions were selected to meet hardware and software requirements:

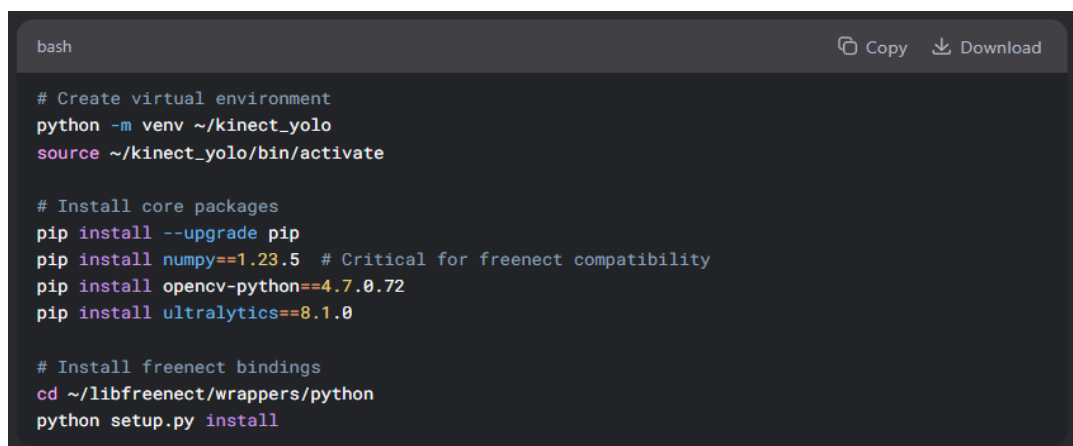
NumPy v1.23.5 was used to maintain compatibility with the data type constraints required by the `freenect` library.

OpenCV v4.7.0.72 was selected to align with the NumPy ABI and ensure stable image processing.

Ultralytics YOLOv8 v8.1.0 provided a reliable framework for real-time object detection.

The `freenect` Python bindings were installed from source and linked directly to the compiled `libfreenect` driver. This setup enabled the Kinect to capture RGB data in real time and deliver it efficiently to the YOLOv8 inference engine.

This carefully constructed environment ensured stable system behavior and performance throughout development, emphasizing version control, dependency isolation, and robust hardware-software bridging.



```
bash
# Create virtual environment
python -m venv ~/kinect_yolo
source ~/kinect_yolo/bin/activate

# Install core packages
pip install --upgrade pip
pip install numpy==1.23.5 # Critical for freenect compatibility
pip install opencv-python==4.7.0.72
pip install ultralytics==8.1.0

# Install freenect bindings
cd ~/libfreenect/wrappers/python
python setup.py install
```

Figure 7: Virtual environment setup and package installation script for integrating YOLOv8 with Kinect using NumPy, OpenCV, and freenect bindings

4.8 DESCRIPTION OF EACH PHASE

The development of the Intelligent Autonomous Disinfection Sterilizing Robot (iADSRob) was carried out in a series of structured phases, each addressing critical components of the system from concept to implementation. These phases ensured a modular, scalable, and reliable design that met the objectives of autonomous and selective disinfection in indoor environments.

Phase 1: Research and Requirement Analysis

This phase involved identifying the scope, objectives, and constraints of the robot. A comprehensive literature review was conducted to explore existing disinfection robots, navigation strategies (SLAM, AMCL), ROS2 tools, and hardware platforms. The team defined the core functional requirements, such as autonomous movement, selective spraying, dual-mode control, and indoor safety. Hardware compatibility, power constraints, cost-effectiveness, and integration feasibility were key considerations.

Phase 2: Mechanical Design and Prototyping

In this phase, the physical frame of the robot was constructed. The chassis was designed to house major components including the disinfectant and sterilizing tanks, Raspberry Pi 4, Arduino Mega, motor driver, and spray nozzles. 3D-printed parts were created for tank holders, ultrasonic sprayer mounts, and tablet stands. The layout was optimized for balance, airflow, and maintenance accessibility. Attention was given to wheel positioning, spray coverage angles, and the mounting of the RPLiDAR sensor for optimal field-of-view.

Phase 3: Hardware Integration

This phase focused on connecting and testing all electronic components. The Raspberry Pi 4 served as the main processor, interfacing with the Arduino Mega via USB for low-level motor and relay control. The RPLiDAR A1 sensor was integrated to provide real-time LiDAR scan data. HC-05 Bluetooth modules enabled wireless manual control. Custom power distribution circuits, including a step-down buck converter and inline fuses, were implemented to ensure electrical protection and component isolation.

Phase 4: Software Development and ROS2 Integration

ROS2 Humble was selected as the software framework for real-time control and navigation. Nodes were implemented for SLAM Toolbox (mapping), AMCL (localization), and Nav2 (path planning and control). Custom ROS2 nodes were created for serial communication with the Arduino and for Bluetooth command parsing. The `robot_state_publisher`, `robot_localization`, and TF broadcasting were configured to maintain spatial consistency. Safety timers, topic monitoring, and graceful shutdown scripts were also included for fault tolerance.

Phase 5: Autonomous and Manual Mode Implementation

To support flexible operations, dual-mode functionality was implemented. In autonomous mode, the robot uses the full ROS2 stack for map-based navigation and obstacle avoidance. In manual mode, Bluetooth commands are interpreted by the Arduino for direct control. A software flag toggles between modes based on launch parameters, allowing safe switching. Safety interlocks were added to prevent conflicting commands from being processed concurrently.

Phase 6: Testing, Simulation, and Optimization

Extensive testing was performed in both simulated and real environments. Gazebo and RViz2 were used for simulating environments, visualizing TF frames, and testing SLAM performance. Real-world tests focused on disinfection coverage, localization stability, and communication latency. Parameters were tuned to optimize path planning speed, SLAM resolution, and power usage. Failure points were logged and addressed in iterative revisions.

Phase 7: Documentation and Final Demonstration

All design steps, diagrams, software configurations, and testing results were documented in this report. A live demonstration was conducted to showcase the robot's ability to autonomously navigate an indoor area, selectively spray designated regions, and respond to manual commands. Visual feedback was displayed on the mounted tablet, including tank levels and control status, completing the presentation of a fully functional proof-of-concept.

4.9 DESIGN ISSUES & LIMITATION

Throughout the development of the autonomous disinfection robot, several technical challenges and system limitations were identified and addressed. One of the primary challenges was maintaining real-time performance on the Raspberry Pi 4. Although the Raspberry Pi is a powerful embedded system for its class, executing simultaneous processes such as SLAM Toolbox, Nav2 navigation stack, RPLiDAR data acquisition, and Bluetooth command handling imposed a significant load on CPU and memory

resources. This occasionally resulted in latency spikes and navigation jitter. These issues were partially mitigated by optimizing node launch configurations, reducing SLAM scan resolution, adjusting Nav2 planner frequencies, and offloading non-critical processes into isolated launch groups to preserve real-time responsiveness.

Bluetooth communication also presented limitations, particularly in environments with dense architectural features or high electromagnetic interference. The HC-05 module, while easy to implement, exhibited reduced signal reliability and range in these conditions, sometimes resulting in lost or delayed commands. To improve remote control reliability, future iterations of the system may adopt Wi-Fi-based communication using MQTT or WebSocket protocols, which offer more stable connectivity and encryption support, and are more compatible with smartphone apps designed for industrial control.

Power management emerged as another critical concern, especially when both humidifiers operated simultaneously under high load. Power spikes and potential thermal buildup were addressed by integrating a relay driver circuit with built-in current protection, spacing components to enhance airflow, and adding ventilation slits to the robot's chassis for passive heat dissipation. Despite these improvements, peak current draw still approached the upper limit of the power supply during full-load operation.

Finally, the system does not currently meet any external safety or medical-grade disinfection certifications. As a result, while the robot performs reliably in indoor environments such as offices or educational facilities, its use in clinical or hospital-grade settings requires additional oversight or certification. Nonetheless, the platform demonstrates a robust and modular foundation, providing a strong proof-of-concept that can be extended for commercial deployment with targeted upgrades in hardware, compliance, and connectivity.

4.10 SAFETY DESIGN AND POWER MANAGEMENT

Although the disinfection robot has not been certified under any formal regulatory standards for sterilization or medical-grade safety, its design strongly emphasizes internal safeguards, operational reliability, and fault tolerance. The system incorporates both hardware- and software-level protection mechanisms to ensure safe use in indoor environments such as clinics, offices, or laboratories. At the hardware level, the power

distribution system is carefully regulated to isolate critical subsystems and prevent damage from power anomalies. The robot operates on a 12V 5A DC power supply, which directly powers high-current components such as the motors and relay-driven humidifiers. A dedicated buck converter steps down the voltage to a stable 5V output suitable for powering the Raspberry Pi, ensuring protection against overvoltage and noise interference. All major circuits are routed through inline fuses to mitigate the risk of short circuits, and overcurrent protection is implemented on the motor driver side to prevent electrical overload during sudden directional changes or mechanical obstructions.

On the software side, several layers of safety logic are embedded in both the ROS2-based control stack and the Arduino firmware. For example, a navigation watchdog monitors the velocity command stream on the `/cmd_vel` topic, and if no new command is received within a two-second window, the Arduino automatically stops the motors to prevent runaway behaviour in case of communication loss or software crashes. The Bluetooth manual control interface includes lockout conditions that restrict relay activation to prevent accidental humidifier triggering from unintended inputs. Additionally, each humidifier is governed by a runtime limit—enforced by software timers on the Arduino—to reduce the risk of overheating or prolonged exposure in a single area, especially important for safety in enclosed spaces. A safe shutdown procedure is also incorporated in the form of a Bash script that flushes serial buffers, terminates active ROS nodes, and unmounts storage devices before initiating a controlled system power-down, ensuring hardware longevity and data integrity.

Collectively, these protective measures allow the robot to operate autonomously for extended periods without requiring direct supervision, while minimizing the risks of mechanical failure, electrical hazards, or unsafe behaviour. The safety architecture enhances user trust and makes the platform viable for deployment in public or semi-public indoor settings, even in the absence of formal safety certification.

4.11 DUAL-MODE OPERATION: AUTONOMOUS AND MANUAL CONTROL

A core architectural feature of the disinfection robot is its ability to operate in dual modes: autonomous navigation using ROS2 and manual control via Bluetooth. This hybrid capability significantly enhances the robot’s versatility, enabling it to perform structured,

map-based disinfection routines while also allowing for manual intervention in complex or unexpected scenarios. In autonomous mode, the robot leverages ROS2's full navigation stack, including SLAM Toolbox or AMCL for localization and Nav2 for path planning and control. During this mode, target waypoints can be set, and the robot autonomously computes and executes optimal paths, dynamically avoiding obstacles based on LiDAR data.

Manual mode, in contrast, bypasses the ROS2 navigation pipeline and instead activates a lightweight Bluetooth control interface. This interface relies on an HC-05 Bluetooth module connected to the Arduino Mega, which interprets user-issued commands from a mobile app to control motion and relay-based actuation. To switch between modes, a runtime software flag is toggled—either through a launch parameter or a command-line argument during boot. This flag is monitored by a custom node or the Arduino firmware, ensuring that command streams from Nav2 are ignored when manual mode is active. Safety interlocks are implemented to prevent both autonomous and manual inputs from being processed simultaneously, thereby avoiding actuator conflicts that could damage hardware or result in erratic behavior.

This dual-mode design not only supports autonomous disinfection in known environments but also empowers operators to conduct on-the-fly interventions, navigate unmapped spaces, or perform localized cleaning without requiring remapping or code changes. Such flexibility is especially valuable in real-world settings like hospitals, schools, and offices, where unpredictable human activity, furniture changes, or temporary obstructions can compromise fully autonomous navigation. The ability to seamlessly switch between modes ensures both operational robustness and user-centered adaptability.

5 PROJECT SIMULATION AND PERFORMANCE EVALUATION

5.1 MAPPING AND LOCALIZATION WITH SLAM TOOLBOX

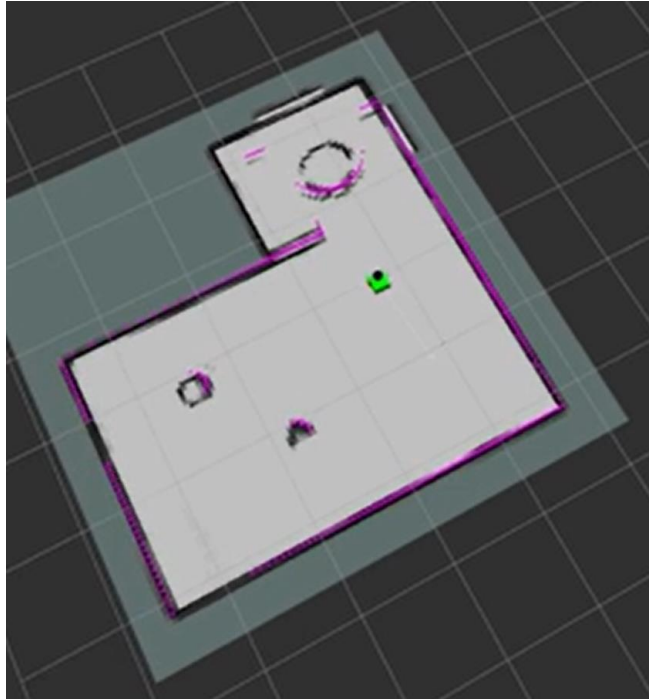


Figure 3.2: Generated 2D occupancy grid map using SLAM Toolbox, enabling real-time mapping and localization for autonomous indoor navigation.

Autonomous navigation in indoor robotics relies fundamentally on the robot's ability to understand its environment and determine its position within that environment. This dual process is accomplished through Simultaneous Localization and Mapping (SLAM). SLAM enables a robot to construct a spatial map of an unfamiliar environment while simultaneously estimating its location within that map. This process is particularly critical in GPS-denied environments such as hospitals, office spaces, and laboratories.

In this project, the SLAM Toolbox is utilized as the core SLAM framework due to its strong integration with ROS2, lightweight computational profile, and robust feature set.

The toolbox operates in two distinct modes: online and offline. In online mode, the robot dynamically builds a map during active exploration. This mode is ideal during initial deployment phases or in spaces with changing layouts. In contrast, offline mode allows the robot to reuse a previously saved map, focusing exclusively on localization. This reduces CPU consumption and enhances runtime stability, especially in familiar or fixed environments.

SLAM Toolbox incorporates advanced techniques such as pose-graph optimization and loop closure detection. As the robot moves and collects laser scan data, it may accumulate drift due to odometry inaccuracies or sensor noise. Pose-graph optimization helps to refine the robot's estimated trajectory by adjusting previous poses to minimize this error. When the robot revisits known areas, loop closures are detected and integrated into the graph, correcting positional drift and reinforcing map accuracy.

The system is designed to support multi-threaded execution, enabling concurrent processing of sensor data, pose estimation, and optimization routines. This architectural design takes full advantage of the Raspberry Pi 4's quad-core processor, ensuring real-time performance without introducing significant latency. This is particularly valuable when operating in large or cluttered environments where fast and reliable processing is essential for decision-making.

For data alignment across subsystems, SLAM Toolbox publishes standard TF2 (transform) frames in ROS2—specifically, `/map`, `/odom`, and `/base_link`. These coordinate frames are foundational for inter-node communication and spatial referencing across the entire system. Navigation modules such as the Nav2 stack and localization nodes like AMCL rely on the transform tree to interpret spatial relationships correctly, ensuring accurate movement, sensing, and feedback.

The toolbox also provides a broad set of configuration parameters that allow for extensive tuning. These include map resolution, maximum range thresholds, keyframe spacing, loop closure sensitivity, and optimization intervals. By adjusting these parameters, the system can be optimized for a variety of scenarios—ranging from dense, narrow corridors to wide, open-plan areas. This flexibility ensures consistent performance across different deployment sites.

To aid in development, testing, and diagnostics, SLAM Toolbox integrates with visualization tools such as RViz2. Through this interface, developers can observe LiDAR scan overlays, pose trails, loop closures, and map evolution in real time. This visual insight accelerates the debugging process and allows for more intuitive parameter tuning based on observed robot behavior.

SLAM Toolbox thus serves as the backbone for spatial awareness in this project, providing a stable and configurable SLAM solution that integrates seamlessly with the rest of the ROS2-based autonomy stack.

5.2 NAVIGATION WITH ROS2 NAV2 STACK

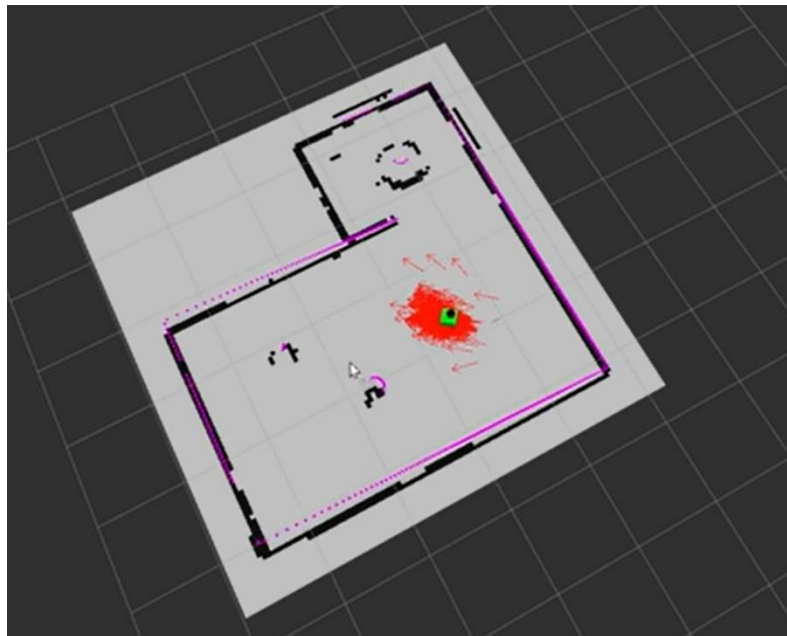


Figure 3.3: ROS2 Nav2 Stack in action, showing global and local path planning for autonomous navigation within a mapped environment.

The navigation capabilities of the disinfection robot are built upon the ROS2 Navigation Stack, widely known as Nav2. This modular and extensible framework forms the core of autonomous motion in ROS2-based systems, offering a complete solution for real-time path planning, obstacle avoidance, and goal-directed behavior. Nav2 serves as the decision-making engine that enables the robot to move safely and efficiently through mapped environments without human intervention.

At the foundation of Nav2 is a set of coordinated subsystems that handle various stages of navigation. The global planner—such as NavFn, SmacPlanner2D, or Hybrid-A*—is responsible for generating an optimal path from the robot’s current position to a specified destination. It does so by analyzing the occupancy grid produced by the SLAM Toolbox and constructing a collision-free route. These planners use search algorithms such as A*, Dijkstra’s, or heuristic-based variants to compute the shortest traversable path through free space while avoiding static obstacles.

Once a global path is generated, the local planner takes over. In this system, the Dynamic Window Approach (DWB) is utilized to compute velocity commands that guide the robot along the global path while adapting to real-time conditions. DWB evaluates a range of possible velocity samples within a dynamic window and scores them based on criteria such as proximity to obstacles, alignment with the global path, and smoothness of motion. The trajectory with the highest cumulative score is selected and executed. This allows the robot to maneuver through tight corridors, around furniture, and past humans with reactive behavior, even in highly dynamic environments.

To support both global and local planning, Nav2 utilizes costmaps, which are layered grid representations of the robot’s surroundings. The global costmap stores a broad view of static obstacles based on the SLAM-generated map, while the local costmap is continuously updated using real-time sensor data, particularly from the RPLiDAR. These maps include inflation zones around obstacles, ensuring that the robot maintains a safe distance and avoids potential collisions due to navigation inaccuracies or sensor noise.

Nav2’s flexibility is one of its greatest strengths. All components are configurable through YAML parameter files, allowing fine-tuning of planner behaviors, costmap resolution, sensor sources, recovery strategies, and controller gains. Additionally, the launch system in ROS2—based on Python launch files—allows developers to compose modular navigation stacks tailored to different robot platforms or deployment scenarios. In this project, for instance, conservative tuning parameters are used to prioritize smooth and cautious motion, appropriate for indoor settings with high human presence and fragile objects.

Another key feature of Nav2 is its support for behavior trees as a decision-making model. Rather than relying on rigid finite state machines, Nav2 uses XML-defined behavior trees

to manage high-level logic such as goal reaching, replanning, recovery from failure, and waypoint navigation. This makes the system highly extensible—custom nodes can be added to handle disinfection events, manual override triggers, or docking procedures without disrupting the base planner architecture.

In this project, Nav2 is also configured to support multi-point waypoint missions, which are crucial for structured cleaning or disinfection routines. A predefined set of waypoints—corresponding to critical locations such as room centers, corridor intersections, and entry points—can be issued as a mission. The robot sequentially visits each waypoint, executing disinfection routines at each location. The waypoint following behavior is also integrated with recovery actions, ensuring the robot can autonomously resume its route if interrupted by dynamic obstacles or localization errors.

In summary, the ROS2 Nav2 stack provides a complete, configurable, and robust navigation solution for the disinfection robot. Through its layered planning architecture, costmap-driven obstacle avoidance, behavior tree coordination, and mission-level waypoint management, it enables fully autonomous, safe, and repeatable navigation in complex indoor environments.

5.3 SIMULATION AND VISUALIZATION: GAZEBO AND RVIZ2

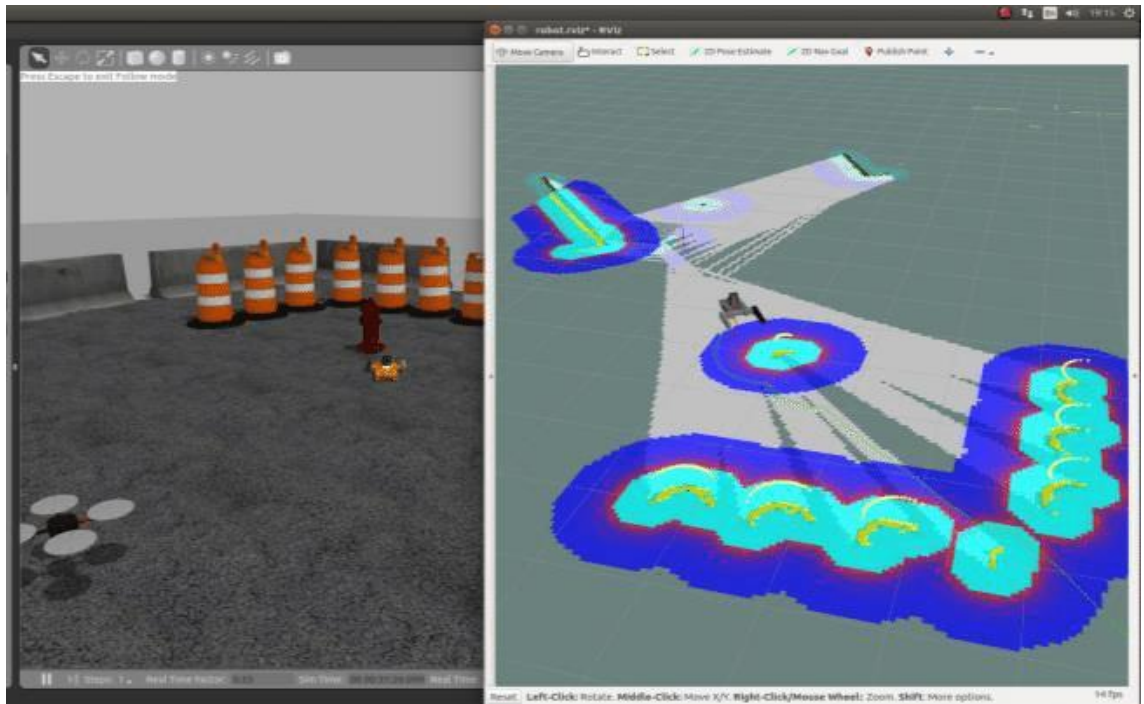


Figure 3.4: Gazebo and RViz2 simulation environments used to test robot behavior, obstacle avoidance, and sensor perception before real-world deployment.

Simulation and visualization are indispensable components in the development of autonomous robotic systems, particularly during the early stages of system integration and testing. In real-world scenarios, testing on physical hardware is often constrained by safety concerns, hardware availability, and the cost of failure. As such, leveraging high-fidelity simulators and visualization tools provides a risk-free and cost-effective pathway to validate both hardware behavior and software logic prior to deployment. This project makes extensive use of two foundational tools in the ROS2 ecosystem: Gazebo for dynamic simulation and RViz2 for real-time visualization.

Gazebo is a powerful open-source 3D robotics simulator that allows the creation of virtual environments with realistic physics, lighting, sensor feedback, and collision models. It is tightly integrated with ROS2 through the `gazebo_ros` packages, enabling full compatibility with ROS nodes, messages, and topics. In this project, Gazebo serves as a virtual testbed where the disinfection robot can operate in digitally constructed hospital

corridors, office layouts, and open-room environments. These simulated environments mirror the conditions under which the robot is expected to perform in reality. Within Gazebo, the complete robotic stack—including the robot's URDF model, differential drive system, LiDAR sensor, and humidifier control interfaces—is simulated. The simulation supports the testing of critical behaviors such as mapping via SLAM Toolbox, path planning via Nav2, and obstacle avoidance through real-time LiDAR data. Actuator commands, velocity feedback, and collision responses are all calculated using Gazebo's built-in physics engine (ODE or Bullet), ensuring accurate reproduction of real-world dynamics.

The use of simulation tools like Gazebo significantly accelerates the development cycle by allowing developers to identify and resolve issues in a controlled and repeatable manner. For instance, navigation bugs, controller tuning errors, or misconfigured transforms can be safely debugged without the risk of damaging hardware. This also enables continuous integration and automated testing, where various robot scenarios can be validated as part of a CI/CD pipeline without human supervision. Furthermore, simulated testing is especially advantageous when dealing with limited access to physical testing environments, such as restricted lab hours or unavailable hospital spaces. It enables rapid iteration on design choices—such as sensor placement, robot geometry, and navigation parameters—by observing their effects directly in the simulated world.

While Gazebo provides a simulation of the physical world, RViz2 complements it by offering a comprehensive 3D visualization platform that reflects the robot's internal state and sensor data in real time. RViz2 is an essential tool for both developers and operators, allowing them to visualize critical information such as real-time LiDAR scan points and their overlays on the generated occupancy grid map, the robot's pose estimation and movement trajectory derived from localization nodes such as AMCL or SLAM Toolbox, navigation paths and goals generated by Nav2's global planner, and the TF (transform) tree that outlines the relationship between coordinate frames including `base_link`, `map`, `odom`, and `laser_frame`. By using RViz2, developers can verify that the robot is interpreting its environment correctly and that all subsystems are functioning as expected. Misalignments in frame transforms, faulty sensor data, or localization drift are immediately observable through RViz2's interface, making it an indispensable tool for debugging.

RViz2 also provides tools for interactive testing, such as 2D pose estimators, goal publishers, and marker displays. Developers and testers can manually assign navigation goals or set robot poses within the visualization interface, which is especially useful for controlled experiments or demonstrations. In addition, RViz2 supports live updates of ROS2 parameters through dynamic reconfiguration, enabling on-the-fly tuning of navigation behaviors, PID controller gains, or SLAM parameters. This interactive workflow bridges the gap between theoretical design and practical implementation, offering a feedback-rich environment that enables data-driven decision-making. It also aids in the development of custom user interfaces or dashboards for monitoring robot health and performance during field deployment.

Together, Gazebo and RViz2 create a closed-loop development environment. Data generated within Gazebo—such as simulated LiDAR scans or wheel encoder data—can be visualized in RViz2 in real time, allowing the developer to evaluate both external robot behavior and internal state estimation simultaneously. This integration ensures that the simulation is not only visually accurate but also consistent with the software stack that will be used in the real robot. For this disinfection robot, the combined use of Gazebo and RViz2 proved essential for validating autonomous navigation in confined indoor spaces, verifying the behavior of Bluetooth/manual override systems, and testing the interaction between ROS2 nodes prior to live deployment.

5.4 AI MODEL PERFORMANCE EVALUATION

Following the training phase, the YOLOv8 model underwent rigorous evaluation using the validation dataset. The key performance metrics achieved were:

Table 1

<i>Metric</i>	<i>Value</i>
Precision	93%
Recall	88%
mAP@0.5	92%
mAP@0.5:0.95	71%
F1 Score	0.90

5.4.1 Class-wise Analysis

- **Light Switches** showed the best F1 score (~0.95) and stable detection across thresholds.
- **Tables and Chairs** also performed exceptionally (~0.94 and ~0.91 F1).
- **Door Handles** had the lowest F1 (~0.78), indicating further training or image diversity is needed.

5.4.2 Confidence Threshold Optimization

F1-Confidence analysis revealed an optimal threshold of 0.42, balancing precision and recall for deployment. Precision and recall vs. confidence graphs were also generated for each class.

5.5 REAL-WORLD TESTING AND DEPLOYMENT

The trained YOLOv8 model was deployed on the robot and tested in live environments including laboratories, classrooms, and corridors. The goals were to assess detection performance, system integration, and real-time operation.

5.5.1 Field Testing

- **Detection Accuracy:** Verified by comparing real-time predictions with expected annotations.
- **Speed:** Achieved real-time processing suitable for mobile operation (~25 FPS).
- **Robustness:** Maintained performance under varying lighting and spatial arrangements.

5.5.2 Integration and Validation

- **System Compatibility:** Model was successfully integrated with the robot's onboard hardware.
- **Real-Time Inference:** Detected objects were used to dynamically adjust the robot's spraying path.
- **Stability:** No crashes or lags were observed during extended operation periods.

5.6 KINECT-BASED TECHNICAL ENHANCEMENTS

This section presents advanced integration techniques used to enhance the Kinect-YOLOv8 system, focusing on spatial alignment of RGB and depth data, performance optimization via multi-threading, and robust handling of sensor data streams. These improvements contribute to more efficient and fault-tolerant operation during real-time object detection tasks.

5.6.1 Depth-RGB Alignment and Threaded Processing

To align depth data with the RGB video stream, the `register_depth_to_rgb()` function was employed, which leverages the Kinect's intrinsic calibration parameters (focal lengths f_x ,

fy, and optical centers cx, cy). This registration enables precise mapping of 3D depth measurements to 2D color imagery, which is crucial for depth-aware object detection.

For performance optimization, the system utilizes a multi-threaded architecture that separates I/O-bound frame capture from CPU-bound YOLOv8 inference. This is implemented using Python's Thread and Queue libraries. As a result, frame drops during intensive processing are minimized, and detection throughput is significantly improved.

5.6.2 Additional Enhancements

- **Depth Visualization:** Converts 11-bit depth data to 8-bit using colormapping techniques, enabling visual feedback of scene depth.
- **Tilt Control:** Allows programmatic control of the Kinect's motorized tilt mechanism ($\pm 27^\circ$) via the `freenect.tilt()` interface.
- **Error Recovery:** Implements automatic reconnection and recovery mechanisms to address USB disconnects or timeout events.

5.6.3 Key Benefits

- Synchronized RGB and depth stream analysis for enhanced 3D perception.
- Up to 30% improvement in processing speed through parallel execution.
- Greater fault tolerance and system stability during continuous operation.

5.6.4 Frame Processing Pipeline Overview

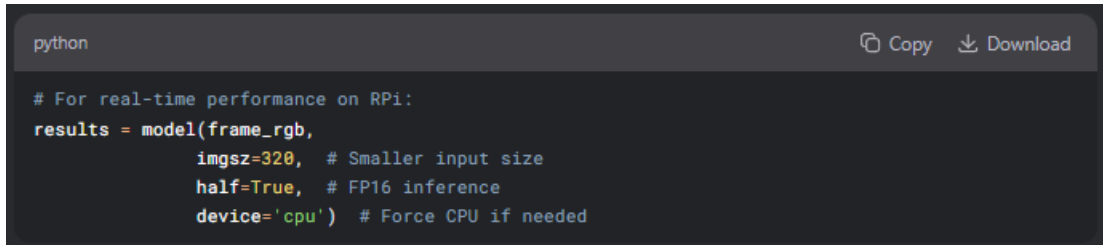
KINECT RGB-D CAPTURE:

- **RGB Video:** Captured at 640×480 resolution @ 30 FPS (BGR format)
- **Depth Stream:** Captured at 11-bit precision (range: 0–2047 mm)

YOLOV8 INFERENCE PIPELINE:

- **Automatic Input Resizing:** Resizes input frames to 640×640 by default
- **Output Format:** Returns a Results object containing:
 - **Bounding Boxes** (xyxy, conf, cls)
 - **Segmentation Masks**
 - **Keypoints** (for pose estimation)

5.6.5 Performance Optimization:

A screenshot of a code editor window with a dark theme. The title bar shows 'python' on the left and 'Copy' and 'Download' icons on the right. The code is a Python snippet for YOLOv8 inference, with comments explaining the optimization settings for real-time performance on a Raspberry Pi. The code uses the 'model' function with parameters for image size, precision, and device.

```
python                                                                    Copy Download

# For real-time performance on RPi:
results = model(frame_rgb,
                imgsz=320, # Smaller input size
                half=True, # FP16 inference
                device='cpu') # Force CPU if needed
```

Figure 8: Code snippet demonstrating YOLOv8 inference configuration optimized for real-time performance on Raspberry Pi using reduced image size and FP16 precision

6 BUSINESS MODEL

6.1 BUSINESS MODEL CANVAS

To assess the commercial potential and deployment feasibility of the Intelligent Autonomous Disinfection Sterilizing Robot (iADSRob), a business model canvas was developed based on the widely adopted framework by Alexander Osterwalder. The canvas helps identify the key elements involved in delivering and sustaining value to customers as shown in Table 2:

Table 2

<i>Business Model Canvas</i>	<i>Details</i>
Key Partners	Hospitals and clinics, healthcare regulators, robotics hardware suppliers, AI/robotics research institutions, sanitation product manufacturers.
Key Activities	R&D, robot manufacturing, software updates, maintenance services, training programs for operators, marketing and customer support.
Value Propositions	Autonomous disinfection in real-time, selective spraying reduces waste, contactless sterilization in high-risk areas, adaptable dual-mode control, low-cost open-source platform.
Customer Relationships	B2B support contracts, on-site training, customer care hotlines, remote diagnostics, custom integration services.
Customer Segments	Hospitals, intensive care units, elderly care facilities, public transportation hubs, airports, schools, offices, and cleaning service providers.
Key Resources	ROS2-based software stack, Raspberry Pi/Arduino hardware setup, SLAM & Nav2 navigation, LiDAR sensor, mobile app control interface.
Channels	Direct sales to hospitals and institutions, partnerships with medical equipment distributors, online marketing, exhibitions, robotics conferences.
Cost Structure	Component sourcing and manufacturing, R&D personnel, cloud infrastructure for updates/support, training and deployment costs.
Revenue Streams	One-time sales, extended warranties, maintenance contracts, custom software feature subscriptions, hardware upgrades.

6.2 COMPONENTS OF THE BUSINESS MODEL

The business model for iADSRob is designed to align with both technical innovation and public health needs. Below is a breakdown of its key components and how each contributes to the robot's commercial viability:

1. Customer Segments

The primary customers are healthcare and public facility stakeholders seeking automated, efficient, and contactless sterilization solutions. This includes:

- Hospitals and isolation wards
- Airports and metro stations
- Government buildings
- Elderly homes and schools

2. Value Proposition

iADSRob offers a unique blend of autonomous intelligence, safety-focused disinfection, and cost-efficiency:

- Reduces human exposure to pathogens
- Selective spraying lowers liquid consumption
- Dual-mode (autonomous/manual) increases adaptability
- Affordable due to open-source and off-the-shelf components

3. Revenue Model

Multiple income channels ensure financial sustainability:

- Direct sales of robots
- Custom feature/software integration
- Maintenance and support plans
- Replacements or upgrades (e.g., sensors, batteries)

4. Cost Structure

The costs are divided into development, deployment, and after-sales:

- Hardware components (sensors, Pi, sprayers, LiDAR)
- Assembly labor and QA testing
- Software development and updates
- Packaging and logistics

5. Key Resources

- In-house engineering and software teams
- Partnership with component suppliers (e.g., LiDAR, microcontrollers)
- ROS2 open-source libraries and development tools

6. Channels

iADSRob will be marketed and distributed through:

- Direct outreach to hospitals and government bodies
- Demonstrations at medical tech expos
- Online platforms for open-source robotics

7. Customer Relationship

The project includes:

- Pre-deployment training
- Remote technical assistance
- Firmware/software update service
- Custom UI options for institutional needs

8. Key Partnerships

- Universities and research labs (for pilot studies and optimization)
- Hardware vendors for bulk component pricing
- Cleaning product companies for chemical recommendations

9. Long-Term Vision

The robot can evolve into a platform for smart facility automation. Future services may include air quality monitoring, patrolling, and voice-interactive assistance in clinical settings.

7 CONCLUSION AND FUTURE WORK

7.1 CONCLUSION

The Intelligent Autonomous Disinfection Sterilizing Robot serves as a fully functional, mobile hygiene system leveraging the ROS2 ecosystem for real-time navigation, control, and environment-aware operation. By integrating cost-effective yet powerful hardware—including the Raspberry Pi 4, Arduino Mega, and RPLiDAR A1—with advanced software modules such as SLAM Toolbox, Nav2, and AMCL, the robot is capable of autonomous indoor sterilization and targeted perfuming in GPS-denied environments. Its design supports dual-mode operation, allowing seamless transitions between autonomous navigation and manual Bluetooth-based control. This ensures operational flexibility, particularly in dynamic or partially known environments where human intervention may be necessary.

The system’s modular hardware architecture and open-source software stack enable rapid prototyping and scalability, making it suitable for deployment in diverse indoor scenarios such as healthcare facilities, offices, and public venues. Despite current limitations—such as the absence of formal safety certifications, constrained onboard processing capacity, and Bluetooth range restrictions—the robot functions as a robust proof-of-concept. It highlights the viability of low-cost autonomous disinfection solutions in real-world applications and lays a strong foundation for future enhancements.

In the context of increased demand for autonomous and contactless sanitation technologies post-pandemic, this project exemplifies the potential of open-source robotics frameworks to meet evolving public health needs. Its success not only reflects the maturity of ROS2 as an industrial robotics platform but also underscores the importance of flexible, intelligent systems in ensuring hygiene and safety in modern indoor environments.

7.2 FUTURE ENHANCEMENTS

While the current prototype achieves reliable autonomous navigation, effective disinfection, and dual-mode control capabilities, a range of future enhancements could further increase its functionality, precision, and readiness for deployment in commercial and clinical settings. One significant upgrade under consideration is the integration of UV-C sterilization lamps to complement the existing humidifier-based disinfection mechanism. UV-C offers proven germicidal efficacy and would provide a broader spectrum of sterilization, especially on surfaces not reachable by mist.

Another critical enhancement is the addition of a camera-based object detection module using frameworks such as YOLOv5 or OpenCV. This would enable the robot to perform dynamic obstacle classification, adapt its navigation strategy in real-time, and recognize context-specific targets such as human presence or high-touch surfaces. Coupled with the robot's LiDAR-based SLAM system, this would result in a more intelligent and responsive autonomous agent.

To enable remote diagnostics and operator supervision, a real-time web-based interface is proposed. This dashboard would display system metrics including battery levels, relay states, active mode (manual/autonomous), and real-time pose tracking. Built using standard web technologies and interfaced through ROS2 bridges, this feature would allow for safer and more accessible fleet management in multi-robot deployments.

Another critical improvement involves implementing an automatic docking and recharging system. By integrating docking stations with wireless or contact-based charging capabilities and coupling them with visual or fiducial marker detection, the robot could operate in a continuous cycle without human intervention, enabling true 24/7 operation in large facilities.

To support untethered operation, the current plug-in power model would be replaced with a high-capacity lithium-ion battery pack featuring protection circuitry and voltage regulation. This change would improve mobility and extend operational runtime while maintaining power supply stability across all subsystems.

Additionally, NFC or QR code-based room identification modules could be deployed throughout the workspace to provide contextual awareness. By reading these markers, the

robot could autonomously adjust its behavior—such as changing disinfection intensity, navigation constraints, or humidifier usage—based on the zone it enters.

Finally, to transition the system from a functional prototype to a medically deployable unit, efforts would be made to ensure compliance with relevant international standards such as IEC 60601 for medical electrical equipment and ISO 13485 for quality management. These upgrades, made possible by the ROS2-based modular architecture and open hardware framework, would transform the platform into a scalable, compliant, and industrial-grade disinfection solution.

REFERENCES.

- [1] J. U. Duncombe, "Infrared navigation—Part I: An assessment of feasibility," *IEEE Trans. Electron Devices*, vol. ED-11, pp. 34–39, Jan. 1959.
- [2] UVD Robots, "UVD Robots – Autonomous disinfection robots," [Online]. Available: <https://www.uvdrobots.com/>
- [3] Xenex Disinfection Services, "LightStrike Germ-Zapping Robot," [Online]. Available: <https://www.xenex.com/lightstrike/>
- [4] Tru-D SmartUVC, "The Power of One Placement," [Online]. Available: <https://tru-d.com/>
- [5] M. Cadena et al., "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age," *IEEE Trans. Robot.*, vol. 32, no. 6, pp. 1309–1332, Dec. 2016.
- [6] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," *IEEE Trans. Robot.*, vol. 33, no. 5, pp. 1255–1262, Oct. 2017.
- [7] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *arXiv preprint arXiv:1804.02767*, 2018. [Online]. Available: <https://arxiv.org/abs/1804.02767>
- [8] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.
- [9] ClearPath Robotics, "Husky Unmanned Ground Vehicle," [Online]. Available: <https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/>
- [10] Mobile Industrial Robots (MiR), "MiR100 – Autonomous Mobile Robot," [Online]. Available: <https://www.mobile-industrial-robots.com/en/mir100/>
- [11] Xenex Disinfection Services, "LightStrike Germ-Zapping Robot," [Online]. Available: <https://www.xenex.com/lightstrike/>
- [12] UVD Robots, "UVD Robots – Autonomous disinfection robots," [Online]. Available: <https://www.uvdrobots.com/>
- [13] Tru-D SmartUVC, "The Power of One Placement," [Online]. Available: <https://tru-d.com/>
- [14] Sky News Arabia, "بالفيديو.. روبوت ذكي لتعقيم المسجد الحرام وتوزيع ماء زمزم," *Sky News Arabia*, Oct. 2021. [Online]. Available: <https://www.skynewsarabia.com/varieties/1452103>

- [15] International Society of Automation (ISA), “Autonomous disinfecting robots join the front line,” *ISA InTech*, May–June 2020. [Online]. Available: <https://www.isa.org/intech-home/2020/may-june/departments/autonomous-disinfecting-robots-join-the-front-line>
- [16] SMP Robotics, “Autonomous disinfection robot S8.2-TF,” [PDF]. [Online]. Available: <https://smprobotics.com/wp-content/uploads/2020/05/desinfection-robots-s82.pdf>
- [17] Tech Xplore, “Hospital sanitation with autonomous robots combining UV and wipes,” *TechXplore*, May 2025. [Online]. Available: <https://techxplore.com/news/2025-05-hospital-sanitation-autonomous-robots-uv.html>
- [18] A. K. Mishra et al., “UAV based Intelligent Disinfection Robot for Indoor Environments,” *arXiv preprint arXiv:2108.11456*, 2021. [Online]. Available: <https://arxiv.org/abs/2108.11456>
- [19] Open Robotics, “ROS 2 Documentation – Humble Hawksbill,” [Online]. Available: <https://docs.ros.org/en/humble>
- [20] ROS Navigation Working Group, “Navigation2 (Nav2) Stack,” [Online]. Available: <https://navigation.ros.org/>
- [21] S. Macenski, “SLAM Toolbox for Lifelong Mapping and Localization in ROS2,” GitHub repository, [Online]. Available: https://github.com/SteveMacenski/slam_toolbox
- [22] R. Santos and S. Silva, “Arduino Bluetooth Control with HC-05,” Random Nerd Tutorials, [Online]. Available: <https://randomnerdtutorials.com>
- [23] Slamtec, “RPLIDAR A1 – 360° Laser Scanner,” [Online]. Available: <https://www.slamtec.com/en/Lidar/A1>
- [24] Raspberry Pi Foundation, “Raspberry Pi Documentation,” [Online]. Available: <https://www.raspberrypi.com/documentation>
- [25] Open Source Robotics Foundation, “Gazebo – Robot Simulation Made Easy,” [Online]. Available: <https://gazebosim.org/>