

UNDERGRADUATE FINAL YEAR PROJECT REPORT

Department of Electronics Engineering

NED University of Engineering and Technology



PureZone: Autonomous UV and Fog Disinfection Rover

Group Number:

Batch: 2020 – 2024

Group Member Names:

Syeda Aiman Fatima

[EL-20001]

Amal Azeem Khan

[EL-20011]

Muhammad Abdul Ahad

[EL-20136]

.....
Dr. Ghous Baksh Narejo

Co-chairman / Professor

Project Advisor

© NED University of Engineering & Technology. All Rights Reserved – [February,2024]

Author's Declaration

We declare that we are the sole authors of this project. It is the actual copy of the project that was accepted by our advisor(s) including any necessary revisions. We also grant NED University of Engineering and Technology permission to reproduce and distribute electronic or paper copies of this project.

Signature and Date

Signature and Date

Signature and Date

.....

.....

.....

Syeda Aiman
Fatima

Amal Azeem
Khan

Muhammad Abdul
Ahad

EL-20001

EL-20011

EL-20136

aimanfatima@
hotmail.se

amalazeemkhan07@
gmail.com

smabdulahad03@gmail.com

Statement of Contributions

- **Syeda Aiman Fatima [EL-20001]** has made central contributions to our autonomous rover project, particularly through her work on the composition, planning, and building up of the UV light system. She has also overseen the integration of the fog generation system, ensuring its effective performance. In addition, she worked on the movement system. Additionally, Aiman played a key role in reviewing and refining the project's documentary. Her extensive involvement in these areas has been crucial to the project's progress and success.
- **Amal Azeem Khan [EL-20011]** played a crucial role in the project, with a primary focus on documentation. She was responsible for overseeing and crafting multiple chapters of the comprehensive project report, ensuring detailed and accurate reporting. In addition to her documentation efforts, Amal equally contributed to the development and implementation of both the UV light and fog generation systems. Her involvement also extended to the overall physical structure of the project, where she contributed significantly to its design and integration. Her multifaceted contributions have been essential to the project's success.
- **Muhammad Abdul Ahad [EL-20136]** held the responsibility for the design, planning and construction of the rover more specifically the autonomous and manual control of the rover, mapping and navigation, motor control and structure design.

Executive Summary

In response to the heightened need for cleanliness in shared spaces exacerbated by the Covid-19 pandemic, this project introduces an innovative solution: the Autonomous UV Disinfection Robot. Traditional cleaning methods have proven inadequate, especially in high-risk environments like restaurants. The project aims to redefine cleanliness standards by developing an autonomous robot using UV lamps for regular surface disinfection.

The background material highlights the deficiencies of conventional sanitation methods and the imperative for innovative solutions. The primary goal is to create a revolutionary approach to achieving higher sanitation standards in communal areas.

The project's objectives include designing a robot adaptable to various contexts, thorough surface disinfection using UV light technology, and creating a highly autonomous robot capable of efficient navigation and table identification.

The methodology unfolds through distinct stages, including meticulous consideration of physical characteristics, sensors, and Lidar technology in the design phase. The subsequent manufacturing phase incorporates UV lamps, power, and motor circuits, while the development of autonomous navigation algorithms ensures accurate and secure mobility.

In conclusion, this initiative presents a creative solution to sanitation challenges in communal spaces. The Autonomous UV Disinfection Robot has the potential to redefine cleanliness standards comprehensively, contributing to both environmental sustainability and public health.

Acknowledgments

We would like to express our sincere gratitude to Sir Ghous Baksh Narejo, our supervisor, for his unwavering leadership and assistance during this project. His wise criticism, knowledge, and support have been crucial in determining the course and outcome of our work. In addition, we extend our heartfelt appreciation to Sir Mohiuddin Zia from NED's Smart City Lab for his important assistance. His technical expertise and for the access of necessary components, which would have been expensive to obtain otherwise, improved the project's progress. Our ability to operate freely and cooperatively was made possible by the Smart City Lab's accessibility to a supportive work environment. We sincerely appreciate their assistance, which has greatly improved the project's overall caliber and result.

Table of Contents

Author's Declaration	ii
Statement of Contributions	iii
Executive Summary	iv
Acknowledgments	v
Table of Contents.....	vi
List of Figures	vii
List of Tables	viii
List of Abbreviations.....	ix
List of Symbols	x
United Nations Sustainable Development Goals	xi
Similarity Index Report	xii
Chapter 1 Introduction	1
1.1 Background Information	1
1.2 Significance and Motivation	1
1.3 Aims and Objectives.....	3
1.4 Methodology	4
1.5 Report Outline	7
Chapter 2 Literature Review	8
2.1 Introduction	8
2.2 Research Papers.....	8
2.3 Summary.....	11
Chapter 3 Lidar Based Mapping Using SLAM Algorithm	12
3.1 Introduction	12
3.2 SLAM Algorithm and its Selection	12
3.3 Map Generation using SLAM Toolbox of Nav2: A Failed Attempt.....	13
3.4 Hector SLAM Algorithm	14
3.5 Successful 2D Map Generation Using Hector SLAM: Testing Phase	16
3.6 Mapping Our Targeted Environment (HPCC NEDUET First Floor)	18
3.7 Conclusion.....	19
Chapter 4 Autonomous Navigation using Stacks.....	18
4.1 Introduction.....	20
4.2 Navigation using NAV2 Stack	20
4.3 Testing the NAV2 stack on simulated environment	21
4.4 Navigating with ROS Navigation Stack: Current Developments	24
4.5 Conclusion.....	24
Chapter 5 Integrated Systems: Motor Control, Drivers and Structural Foundations	
5.1 Introduction	25
5.2 Motor Driver Selection and Overview	25
5.3 Raspberry Pi Integration.....	26
5.4 Hardware Connections	27
5.5 3D Model using Blender App	29
5.6 Testing and Calibration	29
5.7 Code Explanation	30

5.8 Conclusion	31
Chapter 6 Enhanced Disinfection: Fog Module and UV Lights Integration	
6.1 Introduction	32
6.2 Fog Disinfection Module: Design and Operation.....	32
6.3 UV Lights: Types and Mechanisms of Disinfection.....	33
6.4 Integration of Fog Module and UV Lights.....	33
6.5 Efficacy and Benefits of Combined Disinfection	34
6.6 Safety Measures and Considerations	34
6.7 Conclusion.....	35
Chapter 7 Transitioning to Final Phase- System Architecture & Core Functionalities	
7.1 Introduction	36
7.2 ROS Topic Installation.....	36
7.3 Hector SLAM: Building Map of the world	38
7.4 ROS Navigation Stack.....	40
7.5 Transform Tree	42
7.6 Additional components.....	44
7.7 Conclusion.....	45
Chapter 8 System Integration [Hardware Implementation]	
8.1 Introduction	46
8.2 Challenges with Raspberry Pi.....	46
8.3 Transition to Jetson Nano	47
8.4 Integration of sensors & software.....	49
8.5 Software Stack.....	51
8.6 Navigation Algorithm.....	53
8.7 Conclusion.....	55
Chapter 9 Implementation Details	
9.1 Introduction	56
9.2 Hardware Integration.....	56
9.3 Motor Driver Control	57
9.4 Interfacing Jetson Nano and Arduino UNO.....	58
9.5 Block Diagram.....	59
9.6 UV Disinfection System	60
9.7 Fog Disinfection System.....	61
9.8 Integration & Control System.....	62
9.9 Exploration Server and Bash file Integration.....	63

9.10 Conclusion.....	63
Chapter 10 Applications of the Autonomous UV Disinfection Robot	
10.1 Introduction.....	64
10.2 Application in various sectors.....	64
10.3 Autonomous Mapping.....	64
10.4 Autonomous Navigation	65
10.5 Technical Aspects	66
10.6 Wireless Communication	67
10.7 Conclusion	67
Appendix Arduino Control for Motor Control	68
References.....	75

Table of Figures

Figure. 1: “Power Supply Circuit”	5
Figure. 2: “Motor Circuit”	5
Figure. 3: “Navigation stack for robot Autonomous mode”	9
Figure. 4: “Smart city lab 2D environment map (<i>top-down view</i>).....	17
Figure. 5: “HPCC 1 st floor Corridors 2D environment map (<i>top-down view</i>)”	18
Figure. 6: “Simulated gazebo environment”	21
Figure. 7: “Simulated Environment 2D Map (top-down view)”	22
Figure. 8: “Process of navigating the simulated robot”	23
Figure. 9: “Connections for Manual Movement of Robot using WIFI”.....	26
Figure. 10: “Connections for Autonomous Movement”	27
Figure. 11: “Pure Zone Rover base structure (BETA)”	28
Figure. 12: “3D Model of Pure Zone Rover”.....	29
Figure. 13: “Ultrasonic Mist Maker”	32
Figure. 14: “The UV lamp disinfection Robot”	33
Figure. 15: “Pure Zone Rover structure with UV Lights and Fog Disinfection Module”	34
Figure. 16: “The ultraviolet spectrum and its applications”	35
Figure 17: “ROS Topic List ”	37
Figure 18: “Hector SLAM Package in ROS”	38
Figure 19: “ RP Lidar AM27 Model”	38
Figure 20: “Scan Matching between two laser scans”	39
Figure 21: “ROS Navigation Framework”	40
Figure 22: “Transform Tree”	43
Figure 23: “RQT Graph”	43
Figure 24: “Exploration server”	44
Figure 25 : “ROS Bash File”	45
Figure 26: “Jetson Nano vs Raspberry pi”	46
Figure 27: “GUI of Jetson Nano”	48
Figure 28: “RP Lidar Package”	50
Figure 29: “Realsense Tracking Model”	50
Figure 30: “Realsense Tracking Model package”	51
Figure 31: “Navigation Examples”	53

Figure 32 : “ Command Velocity Package”	55
Figure 33: “Teleop Twist Command on ROS”	57
Figure 34: “Block Diagram”	59

List of Abbreviations

ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
UV	Ultra Violet
LIDAR	Light Detection and Ranging
NAV2	Navigation-2
HPCC	High Performance Computing Centre
DC	Direct Current
GPIO	General Purpose I/O
PWM	Pulse Width Modulation

United Nations Sustainable Development Goals

The Sustainable Development Goals (SDGs) are the blueprint to achieve a better and more sustainable future for all. They address the global challenges we face, including poverty, inequality, climate change, environmental degradation, peace and justice. There is a total of 17 SDGs as mentioned below. Check the appropriate SDGs related to the project.

- ☐ No Poverty
- ☐ Zero Hunger
- ☒ Good Health and Well being
- ☐ Quality Education
- ☐ Gender Equality
- ☒ Clean Water and Sanitation
- ☒ Affordable and Clean Energy
- ☐ Decent Work and Economic Growth
- ☒ Industry, Innovation and Infrastructure
- ☐ Reduced Inequalities
- ☐ Sustainable Cities and Communities
- ☒ Responsible Consumption and Production
- ☐ Climate Action
- ☐ Life Below Water
- ☐ Life on Land
- ☐ Peace and Justice and Strong Institutions
- ☐ Partnerships to Achieve the Goals

Similarity Index Report

Following students have compiled the final year report on the topic given below for partial fulfillment of the requirement for Bachelor's degree in Electronic Engineering.

Project Title **PureZone Rover- An Autonomous UV and Fog Disinfection Robot.**

S.No.	Student Name	Seat Number
1	Syeda Aiman Fatima	EL-20001
2	Amal Azeem Khan	EL-20011
3	Muhammad Abdul Ahad	EL-20136

This is to certify that Plagiarism test was conducted on complete report, and overall similarity index was found to be less than 20%, with maximum 5% from single source, as required.

Signature and Date

.....
Dr. Ghous Baksh Narejo

FYDP final report-1.pdf

by Turnitin LLC

Submission date: 30-Jul-2024 09:13PM (UTC+0700)

Submission ID: 2423564322

File name: uploads_1571_2024_07_30_FYDP_final_report-1_6275edb954aac0bf.pdf (3.66M)

Word count: 14063

Character count: 87598

FYDP final report-1.pdf

ORIGINALITY REPORT

4%

SIMILARITY INDEX

4%

INTERNET SOURCES

3%

PUBLICATIONS

4%

STUDENT PAPERS

PRIMARY SOURCES

1

www.supotronix.com

Internet Source

3%

2

Submitted to Higher Education Commission
Pakistan

Student Paper

1%

3

fabacademy.org

Internet Source

<1%

4

github.com

Internet Source

<1%

5

raiith.iith.ac.in

Internet Source

<1%

Exclude quotes On

Exclude bibliography On

Exclude matches < 5 words

Chapter 1

Introduction

1.1 Background Information

The Covid-19 pandemic has drastically changed the landscape of public health, highlighting the pressing need for improved shared space sanitation standards. Restaurants and social places are among the spaces where infectious illness transmission is most likely to occur.[1] Conventional cleaning techniques, which include chemical agents and manual disinfection, frequently fail to provide the essential degree of hygienic conditions and cleanliness to successfully stop the spread of diseases. As the pandemic spread, it became more and more clear that creative solutions were required to get over the shortcomings of the current sanitation strategies. Traditional methods struggle to maintain uniform standards of disinfection, and they also involve significant labor and time commitments.[1]

In light of these challenges, there is a clear demand for innovative solutions that surpass the constraints of current sanitation methods. The creation of an autonomous UV disinfection robot signifies a revolutionary change in meeting this demand. Going beyond the immediate issues posed by the Covid-19 pandemic, this self-governing robot aims to reshape the realm of sanitation in shared spaces, introducing an inventive approach that is both effective and capable of upholding steadfast and stringent disinfection standards.

1.2 Significance and Motivation

1.2.1 Significance of the Project:

The autonomous cleaning robot proposed in this project holds considerable significance by effectively addressing various critical challenges and presenting notable advantages:

The robot's use of UV lights represents an innovative method to achieve elevated sanitation levels. Through systematic disinfection of restaurant tables, it plays a crucial role in minimizing the risk of viral transmission, particularly in the context of the ongoing Covid- 19 pandemic[2]. This contributes significantly to creating a safer and healthier environment for

both patrons and staff. A key contribution of the project lies in the robot's ability to function autonomously. By employing advanced detection mechanisms, it can identify and sanitize unoccupied tables without requiring human intervention. This autonomy not only streamlines cleaning processes but also allows human staff to focus on essential tasks, thereby enhancing overall operational efficiency in public spaces.

The versatility inherent in the robot's design extends its impact beyond the confines of restaurants. Its adaptability makes it well-suited for deployment in various settings, including offices, schools, and healthcare facilities. This broad applicability underscores the project's potential to contribute to maintaining cleanliness and reducing the risk of infections across diverse environments.

A conscientious feature of the autonomous cleaning robot is its hand sanitization system, which operates with minimal water usage. This addresses urgent environmental concerns related to water conservation while ensuring effective germ-killing capabilities. This dual functionality aligns with sustainable practices, making the project not only beneficial for public health but also environmentally responsible.

1.2.2 Motivation:

The motivation behind creating an autonomous UV disinfection robot stems from recognizing the inherent limitations of traditional cleaning methods and the critical importance of maintaining rigorous hygiene standards.[1]. Restaurants, being central hubs for social interaction, demand a level of cleanliness beyond what conventional practices can achieve. The autonomous robot, equipped with cutting-edge technologies like Lidar and a suite of electronic components, seeks to transform the very essence of sanitation.[3]

This project acknowledges that the need for advanced sanitation practices goes beyond the immediate challenges presented by the current pandemic. It anticipates the ongoing threat of infectious diseases and aims to establish a sustainable and efficient solution for maintaining optimal cleanliness in public spaces. Through the integration of state-of-the-art

technologies into its design, the project endeavors to overcome the limitations of traditional cleaning methods, offering a systematic, autonomous, and precise approach to surface disinfection.

Subsequent chapters will extensively review existing literature, providing comprehensive insights into the deficiencies of traditional cleaning methods and the pros and cons of autonomous disinfection technologies. Additionally, these chapters will delve into the specific objectives of the project, elaborate on the complexities of the system architecture, offer a detailed account of the design and development methodology, and explain the implementation of the autonomous UV disinfection robot's sanitation mechanism and electronic components. Collectively, these chapters contribute to a profound understanding of the project's significance and its potential to redefine sanitation practices in public spaces.

1.3 Aims and Objectives

1.3.1 Aims:

The central objective of this project is to spearhead the development of an advanced autonomous cleaning robot. This cutting-edge robot is intricately crafted to excel in various critical facets, contributing significantly to the improvement of cleanliness and hygiene standards, particularly amid the ongoing Covid-19 pandemic.

1.3.2 Objectives:

The project's specific goals are diverse. Firstly, under the umbrella of the Multipurpose Sustainable Sanitation aim, the objective is to design a robot initially intended for restaurants but adaptable to disinfecting tables in various settings. This involves creating a distinctive hand sanitization solution that utilizes fog and UV light at specific points, promoting effective hand hygiene while simultaneously addressing water conservation.

Secondly, in alignment with the Advanced UV Disinfection aim, the objective is to leverage state-of-the-art UV light technology in a robot arm for the thorough disinfection of tables.

This includes implementing and integrating cutting-edge UV light technology to elevate sanitation standards, particularly in the post-pandemic landscape.

Lastly, under the Precision Autonomous Navigation aim, the objective is to create a highly autonomous robot capable of efficient and precise navigation. The specific objective here is to develop and integrate autonomous navigation systems that empower the robot to independently detect and locate unoccupied tables, execute UV disinfection, and autonomously return to its starting position without requiring human intervention.

In essence, the project aspires to transform cleanliness and hygiene standards during the Covid-19 pandemic by accomplishing these objectives and offering an innovative solution to the challenges presented by infectious diseases in communal spaces such as restaurants.

1.4 Methodology

The progression of the project involves several distinct phases, starting with the Design Phase. Within this phase, the physical attributes of the robot, such as its dimensions, configuration, and mass, are established as the foundational elements of its design. Sensors are strategically positioned to ensure precise identification of tables and potential obstacles, with a strong emphasis on reliability. The arm mechanism, crucial for emitting UV light during the table disinfection process, is meticulously designed to prioritize both ergonomics and safety. At the core of this autonomous system lies the pivotal electronic component, the Lidar sensor, which generates a three-dimensional map of the surroundings. This mapping capability is instrumental for enabling autonomous navigation and obstacle avoidance. The Lidar technology serves as the central component, providing the system with the precision and reliability necessary for the development of the autonomous UV cleaning robot.

During the project's initial stage, we will meticulously assemble the robot's chassis, wheels, and motor system, ensuring precise alignment and functionality. To power the robot's chassis, wheels, and motor system, we will integrate the Power supply circuit as depicted in Figure 1.

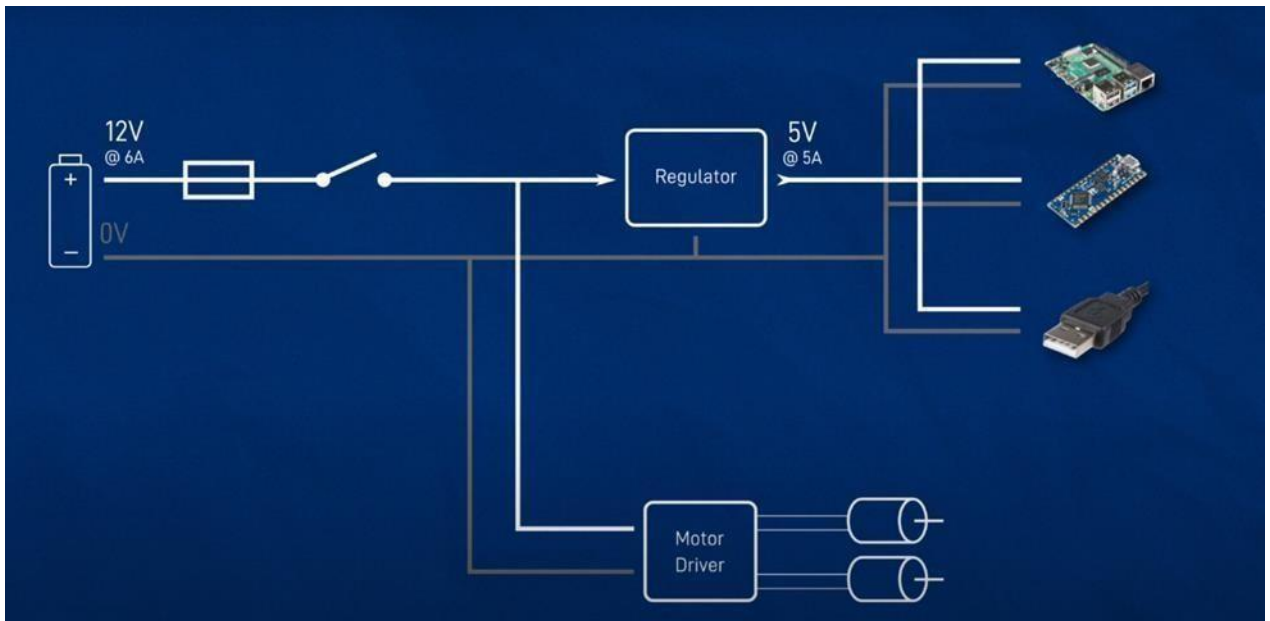


Figure 1 “Power Supply Circuit”

The Motor circuit, presented in Figure 2, will illustrate the connection between the motor, motor driver, and robot controller.

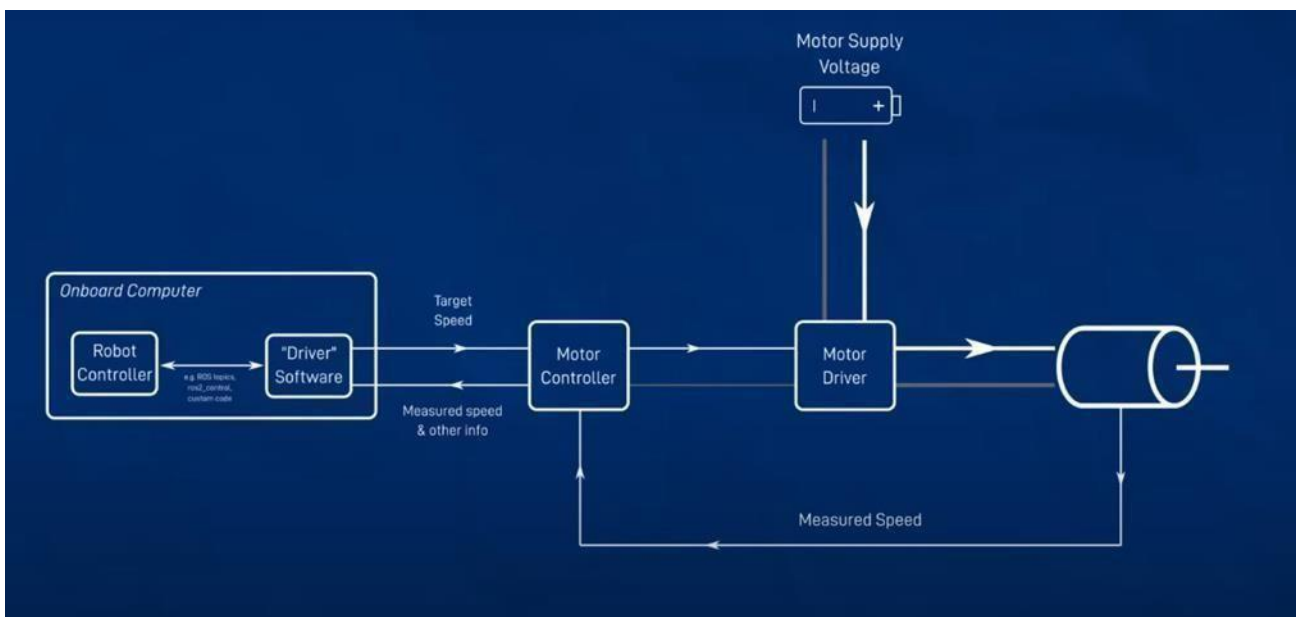


Figure 2 “Motor Circuit”

Following this, the sensors identified during the design phase will be incorporated into the

robot's hardware, necessitating meticulous wiring and calibration to ensure accurate obstacle avoidance. The project will also encompass the integration of UV lights [4] into the robot's arms, with a focus on secure placement, power supply considerations, and safety precautions. This comprehensive approach is designed to enhance the robot's efficiency in disinfecting tables while giving paramount importance to user and device safety.

In the upcoming phase of the project, algorithms will be developed to enable the robot's autonomous navigation within a restaurant setting, accompanied by accurate table detection capabilities. These algorithms play a crucial role in ensuring the robot can navigate independently and identify tables with precision.

Simultaneously, the integration of sensors into the robot's hardware will provide essential data for detecting obstacles in the robot's path. The subsequent step involves implementing obstacle avoidance mechanisms, utilizing the sensor data to navigate around potential obstacles. This ensures the robot's movement is safe, minimizing the risk of collisions.

Additionally, control systems will be created to govern the movement of the robot's arm and regulate the emission of UV light during the table disinfection process. These control systems will be meticulously designed to prioritize precision and reliability, contributing to the overall efficiency of the robot in its crucial tasks while maintaining a strong focus on safety.

1.5 Report Outline

This project seeks to create a self-sufficient cleaning robot tailored for restaurant applications, with a primary focus on disinfecting tables to bolster cleanliness and hygiene amidst the challenges posed by the Covid-19 pandemic. The envisioned robot will possess a compact and lightweight design, incorporating ultrasonic and infrared sensors to identify vacant tables and potential obstacles. To ensure effective sanitization, UV lights will be seamlessly integrated into the robot's arms, strategically positioned at an optimal distance from the table surface. The design incorporates UV lights and a fog disinfection module to clean the surroundings. This approach aims to streamline cleaning procedures, mitigate viral transmission risks, and advocate for environmental sustainability through water conservation strategies.

Chapter 2

Literature Review

2.1 Introduction

This chapter provides an introduction to the extensive body of literature that supports our initiative. Here, we walk through the organized framework that directs the construction of our thesis, highlighting the critical significance that well-written introductions and summaries have. In this section of our literature study, we highlight some key research papers that have had a major impact on the direction of our project. A UV disinfection robot with automatic switching based on human detection is examined in the second study, while UltraBot, an autonomous UV-C disinfection robot, is introduced in the first. In light of the current global health concerns, we want to extract important lessons from these publications that will guide our approach to autonomous robotic solutions for safe and efficient disinfection.

2.2 Research Papers

2.2.1 UltraBot: Autonomous Mobile Robot for Indoor UV-C Disinfection

The paper focuses on the development of an autonomous robot named UltraBot designed to mitigate the transmission of COVID-19 and other harmful pathogens. The research underscores the efficacy of disinfecting surfaces and wearing masks as primary measures against COVID-19, with surface cleaning deemed consistently successful [1], [2].

Autonomous robotic systems have witnessed increased adoption across various industrial domains, encompassing applications like plant disease detection, large-space cleaning and sterilization, and warehouse stocktaking [3], [4], [5]. These robots, capable of disinfecting public areas through tasks like clearing recycle bins and cleaning floors, aim to reduce the spread of infections [6].

The objective of UltraBot is to perform disinfection tasks without resorting to harmful sprays and chemicals, which may leave residues, require prolonged airing, and corrode metal structures. The technology underlying UltraBot is positioned as offering optimal autonomous disinfection performance while prioritizing human safety by minimizing

exposure to UV-C radiation.

The paper delves into the mechanical and electrical structures of UltraBot, along with its low-level and high-level control systems. Experiments conducted on the robot's localization module and optimal routes for UV-C disinfection demonstrate its effectiveness. Notably, the results reveal a significant decrease (94%) in the total bacterial count (TBC) at a distance of 2.8 meters from the robot after 10 minutes of UV-C irradiation.

Autonomous robotic systems have gained popularity in various industrial sectors due to their advantages, such as the ability to use potent disinfectants without endangering humans and consistent effectiveness. A navigation stack of robot Autonomous modes is provided in Fig. 3. The technical solutions for disinfection are categorized based on mobility and disinfectant used. Non-mobile systems include stationary installations with UV-C lamps or UV-based air flow cleaner-recirculatory. However, mobile disinfecting robots face challenges in navigating dynamic, cluttered, and narrow environments.

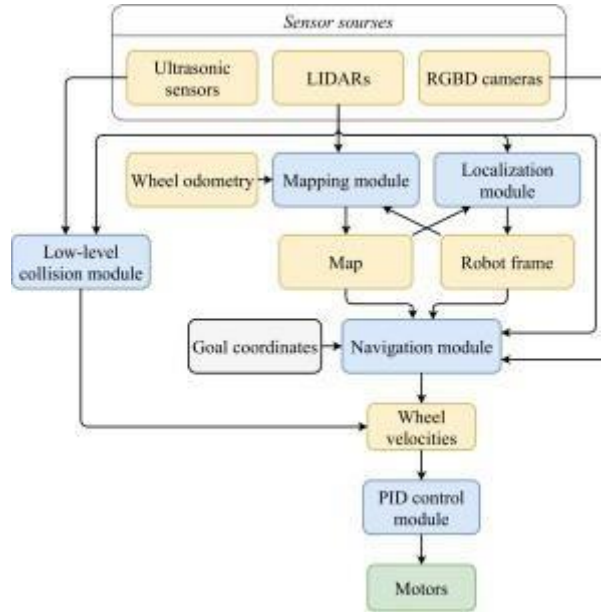


Fig. 3. “Navigation stack for robot Autonomous mode”

Referencing from [18]

2.2.2 “UV Disinfection Robot with Automatic Switching on Human Detection”

In our literature review, we delved into the utilization of UV-C light for disinfection and sterilization, emphasizing its germicidal properties against bacteria and viruses [7]. In

response to the potential harm posed to humans, we found that a UV Robot was designed to autonomously disinfect spaces, equipped with three 20W UV lamps and an embedded system employing Arduino and PIR sensors to detect human or animal presence, ensuring safety [7]. Our exploration aimed to understand the overarching goal of disinfection and sterilization, which is to eliminate microorganisms, thereby mitigating the spread of airborne diseases and infections [8]. UV light emerged as a potent solution, with UV-C radiation, possessing a wavelength of 100-280nm, chosen for its ability to destroy bacteria [9]. We identified the significant potential of UV-C light, especially in the context of sterilization, as a crucial element in combatting novel viruses such as the Coronavirus [10].

In the background section, we uncovered historical context, highlighting the common use of UV light in hospitals during the 1930s and its subsequent significance in the 1950s for air and water treatment, especially in eradicating diseases like TB [17, 19]. We elucidated the mechanism of UV-C light in disinfection, involving DNA rupture and molecular structure damage through photo-dimerization [11, 9]. While acknowledging the potential danger of direct exposure to UV-C radiation for humans, we noted that passing through glass or transparent plastic casing filters out UV-C radiation, allowing only the harmless blue light to pass through [11, 9].

In the methodology section, we detailed the components and functionality of the UV sterilization robot, encompassing PIR sensors, UV lamps, Arduino microcontroller, and IR sensors [12, 13, 14, 15,]. The robot's operation included automatic ON/OFF based on human or animal detection, ensuring safety during the disinfection process [13]. The working process entailed the robot adhering to a predefined path, pausing upon detecting humans or animals, and automatically resuming the disinfection process once the area was clear [16, 17].

2.3 Summary

In conclusion, these research publications have enhanced the literature review we completed for our study. Important details about the mechanical and electrical architecture, control systems, and efficacious disinfection experiments—particularly in terms of bacterial count reduction—of Ultra Bot, an autonomous UV-C disinfection robot, are provided in the first paper on the device. With regard to the second paper, which is about a UV Disinfection Robot with Automatic Switching based on Human Detection, important details regarding the history of UV light, its germicidal qualities, and an extensive process for creating a UV sterilization robot with safety features are covered. These research articles have been extremely important in building our project and our methodology for developing autonomous robotic disinfection solutions. The methods and conclusions from these studies add a great deal to our knowledge and help shape the creation of safe and efficient disinfection techniques to deal with today's pressing global health issues.

Chapter 3

Lidar Based Mapping Using SLAM Algorithm

3.1 Introduction

The crucial topic of choosing a Simultaneous Localization and Mapping (SLAM) algorithm for the Pure Zone Rover—An Autonomous UV disinfection Robot—is explored in this chapter. SLAM is a key component in the vast field of robotics, allowing robots to simultaneously map and traverse new environments. The difficulties encountered the nuances of choosing an appropriate SLAM algorithm for the Pure Zone Rover are the main topics of this chapter. Improving the robotic system's flexibility and autonomy is the aim, and it will also provide light on the difficulties and factors that must be taken into account when making such an important choice.

3.2 SLAM Algorithm and its Selection

In the realm of robotics, Simultaneous Localization and Mapping (SLAM) is an essential idea, especially for developing autonomous systems. It describes the ability of a robot or other technology to simultaneously map an unfamiliar area and instantly determine where it is in relation to that environment. In essence, SLAM allows a robot to independently explore and navigate a foreign environment by continuously adjusting its internal map and fine-tuning its location in response to sensory input. In order to provide a clear and accurate depiction of the environment, this method integrates sensor data, including measurements from cameras, LiDAR (Light Detection and Ranging), and other positional sensors. In order to provide a clear and accurate depiction of the environment, this method integrates sensor data, including measurements from cameras, LiDAR (Light Detection and Ranging), and other positional sensors. Because SLAM algorithms enable autonomous cars and mobile robots to navigate and function in dynamic, unstructured situations without prior knowledge, they are essential to a wide range of applications. The autonomy and adaptability of robotic systems are greatly enhanced by the effective application of SLAM. A crucial stage in the creation of robotic systems, such as the PureZone Rover, is the selection of the (SLAM) algorithm. Selecting the right method for SLAM implementation is essential for efficient navigation and mapping of uncharted territory. During the selection

process, a number of factors are evaluated, including alignment with project-specific needs, computing efficiency, and real-time capabilities. The assessment of computational efficiency involves analyzing the processing speed, memory consumption, and overall complexity of the algorithm to make sure it can function within the limitations of the robot's onboard computing resources. Making decisions instantly requires real-time capabilities, thus selecting an algorithm that can deliver current mapping data quickly is critical.

3.3 Map Generation using SLAM Toolbox of Nav2: A Failed Attempt

In our pursuit of advancing lidar-based mapping within the scope of our FYDP, we undertook the implementation of the SLAM Toolbox, a robust ROS package developed by Nav2. The SLAM Toolbox, designed as a versatile replacement for established SLAM libraries such as gmapping, cartographer, karto, and hector, offered a promising set of features that aimed to address mapping and localization challenges in various environments, including massive and dynamic indoor spaces.

The SLAM Toolbox integrates information from laser scanners in the form of LaserScan messages and TF transforms from odom->base link. One of its key features is the ability to create a 2D map of a given space, providing a comprehensive serialization of both the data and pose-graph of the SLAM map. This feature allows users to reload and continue mapping, localize, merge, or manipulate the map as needed. Furthermore, the toolbox can operate in both synchronous and asynchronous modes, accommodating different operational requirements.

3.3.1 Challenges Encountered:

The SLAM Toolbox, being in its early stages, posed significant technical challenges that impeded its effective integration into our project. The primary issue stemmed from its limited support for real-time mapping using the RP Lidar. Despite our efforts to configure and adapt the toolbox to our requirements, the early-stage development nature of the toolbox hindered seamless integration, particularly in the context of real-time mapping. These technical challenges significantly hampered the robustness and dependability of our autonomous navigation system. Faced with these obstacles, we made a strategic decision to

transition to HectorSLAM, a move crucial for the project's success.

The switch to Hector SLAM played a pivotal role in overcoming technological obstacles and ensuring a smoother integration with our sensor suite. The open-source nature of Hector SLAM and its active community support facilitated quick troubleshooting and customization, significantly enhancing the dependability of our Simultaneous Localization and Mapping (SLAM) implementation.

3.4 Hector SLAM Algorithm

Hector SLAM, developed by the Autonomous Systems Lab at ETH Zurich, is a real-time and open-source Simultaneous Localization and Mapping (SLAM) algorithm primarily designed for robots and unmanned aerial vehicles (UAVs) equipped with lidar sensors.

3.4.1 Working of Hector SLAM:

Hector SLAM operates through a grid-based mapping approach, discretizing the environment into cells that represent the likelihood of occupancy, forming an occupancy grid map. Its core scan matching algorithm refines the robot's pose by aligning consecutive lidar scans, updating position and orientation in real-time as the robot moves. Loop closure detection addresses accumulated errors, correcting the map and refining poses for enhanced accuracy. While initially designed for 2D mapping, Hector SLAM supports optional 3D mapping for a more detailed environment representation. The algorithm dynamically handles moving objects, distinguishing between static and dynamic elements. Integrated with the Robot Operating System (ROS), Hector SLAM ensures compatibility with other ROS packages, simplifying communication in robotic systems. Known for its lightweight design, Hector SLAM excels in real-time performance, making it ideal for applications with limited computational resources.

3.4.1

Hector SLAM (Simultaneous Localization and Mapping) is a reliable system designed for generating maps of environments and determining a robot's location within those maps. particularly effective in real-time scenarios, as it doesn't require odometry data, distinguishing it from many other SLAM systems. Instead, Hector SLAM utilizes high-frequency LIDAR data and, optionally, IMU data.

Sensor Data and Usage

1. LIDAR (Light Detection and Ranging):

- **Purpose:** LIDAR serves as the main sensor for creating 2D occupancy grid maps by measuring distances through laser pulses and their reflections from surrounding objects.
- **Data Used:** The distance measurements, or range data, are converted into coordinates representing points in the environment, which are then used to update the occupancy grid map.
- **Equations Involved:** The positions of the points $S_i(\xi)$ in the map are calculated using the robot's estimated pose ξ (position and orientation). The transformation of scan points $s_i = (s_{i,x}, s_{i,y})^T$ into map coordinates involves rotation and translation:

$$S_i(\xi) = \begin{pmatrix} \cos(\psi) & -\sin(\psi) \\ \sin(\psi) & \cos(\psi) \end{pmatrix} \begin{pmatrix} s_{i,x} \\ s_{i,y} \end{pmatrix} + \begin{pmatrix} p_x \\ p_y \end{pmatrix}$$

- **Explanation:** The LIDAR data provides high-resolution information about the environment's structure. By aligning the LIDAR scans with the existing map, Hector SLAM refines the pose estimate of the robot.

2. IMU (Inertial Measurement Unit):

- **Purpose:** The IMU provides supplementary data regarding the robot's motion, including acceleration and angular velocity, aiding in tracking the robot's orientation and movement between LIDAR scans.
- **Data Used:** The IMU data includes angular rates $\omega = (\omega_x, \omega_y, \omega_z)^T$ and linear accelerations $a = (a_x, a_y, a_z)^T$

- **Position Update:**

$$\dot{p} = v$$

- **Velocity Update:**

$$\dot{v} = R_\Omega \cdot a + g$$

Here, R_Ω relates the angular rates to changes in the Euler angles $\Omega = (\phi, \theta, \psi)^T$, $p = (p_x, p_y, p_z)^T$ is the position, and $v = (v_x, v_y, v_z)^T$ is the velocity. R_Ω is the rotation matrix transforming body frame accelerations to the navigation frame, and g is the gravity vector.

- **Explanation:** The IMU helps in maintaining a continuous estimate of the robot's pose, particularly during fast movements or in situations where LIDAR data alone might not be sufficient for accurate localization.

3.4.2 Integration of Sensor Data in SLAM Process

1. Map Update and Scan Matching:

- The LIDAR data updates the 2D occupancy grid map, representing the environment's structure.
- Scan matching aligns new LIDAR scans with the existing map to refine the robot's pose estimate, minimizing the error function:

$$\xi^* = \arg \min_{\xi} \sum_{i=1}^n [1 - M(S_i(\xi))]^2$$

- This optimization process involves computing the Hessian matrix H and the gradient to find the pose update $\Delta\xi$.

Motion Model and State Estimation:

The IMU data provides continuous updates on the robot's motion, allowing the system to track changes in orientation and velocity. This data is crucial for accurate state estimation, especially in dynamic environments or when the robot is moving rapidly.

Multi-Resolution Map Representation:

Hector SLAM employs a multi-resolution map representation, maintaining maps at different resolutions. This approach helps avoid local minima during scan matching and improves robustness in map creation.

3.4.5 No Requirement for Odometry Data

One of the unique aspects of Hector SLAM is that it does not require odometry data, which is often used in other SLAM systems to estimate the robot's movement based on wheel encoders. This reliance on odometry can be problematic due to issues like wheel slippage and drift, leading to inaccurate pose estimates.

Hector SLAM avoids these problems by relying solely on high-frequency LIDAR data for map updating and scan matching. The system's use of LIDAR data allows for precise localization without the errors introduced by odometry. Additionally, the optional use of IMU data provides further refinement of the pose estimation, especially in the z-axis or under conditions where the LIDAR data alone may not suffice.

By not depending on odometry, Hector SLAM can achieve high accuracy and robustness in a variety of environments, including those with challenging surfaces or conditions that would degrade the quality of odometry data. This makes it particularly suitable for applications such as indoor navigation, exploration, and search and rescue missions, where accurate localization and mapping are critical.

3.5 Successful 2D Map Generation Using Hector SLAM: Testing Phase

We comprehensively analyzed the features of hector slam, we also made sure that hector slam fulfils our mapping criteria and we don't run into problems in which we would spend hours debugging. After careful consideration our first thought was to generate the map of our surrounding environment

3.5.1 Testing the Algorithm:(mapping our surroundings)

We tested the algorithm by mapping our surrounding environment which was the Smart City Lab of NEDUET where the designing of our robot is conducted, So the first 2- dimensional map generated was of the Smart City Lab. The map is shown in the figure 4.

After observing the map, it can be seen that the map shows some black area and the rest of the area is grey, keeping in mind that the black area represents obstacles from where the laser rays from the RP LIDAR are bounced back where as the grey area is the free space from where the laser rays can traverse.

It should also be noted that at the time when the algorithm was running the RP LIDAR was placed at a higher altitude than normal so that minimum number of objects denied the path of the laser rays which can be confirmed by looking at the map.

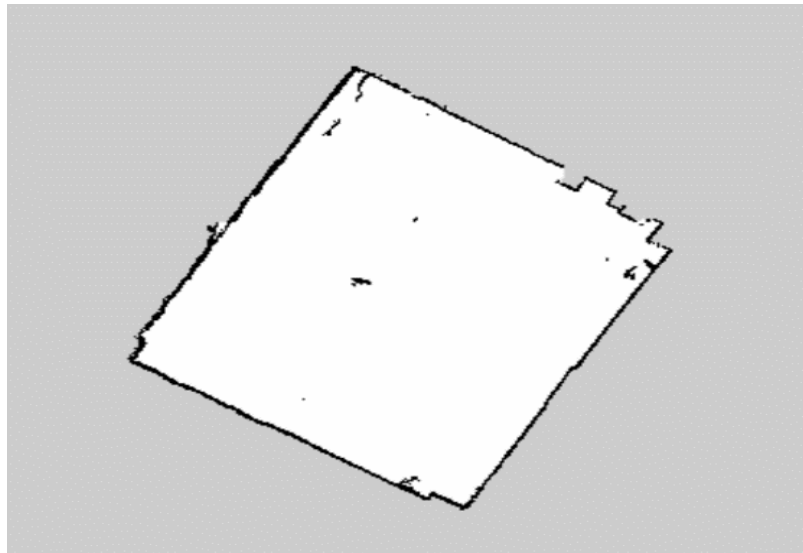


Fig. 4. “Smart city lab 2D environment map (top-down view)”

Now after successfully testing the algorithm we proceeded to generated the map of our targeted environment which is the First floor of HPCC center NEDUET.

3.6 Mapping Our Targeted Environment (HPCC NEDUET First Floor)

After thorough testing and an in depth understanding of the algorithm, it was relatively simple to map our targeted environment (i.e. HPCC NEDUET First Floor). As we know that in order to update the map in real time, the movement of lidar is necessary throughout the environment

keeping in mind that at this point our robot was unable to move remotely. So, when we were in the process of map generation, we had to move the lidar which was connected to the device manually by carrying both the device and lidar in our hands.

By using this approach, we were able to generated the 2-dimensional environment map of 2 of the corridors on the first floor of HPCC which can be shown in figure 5.

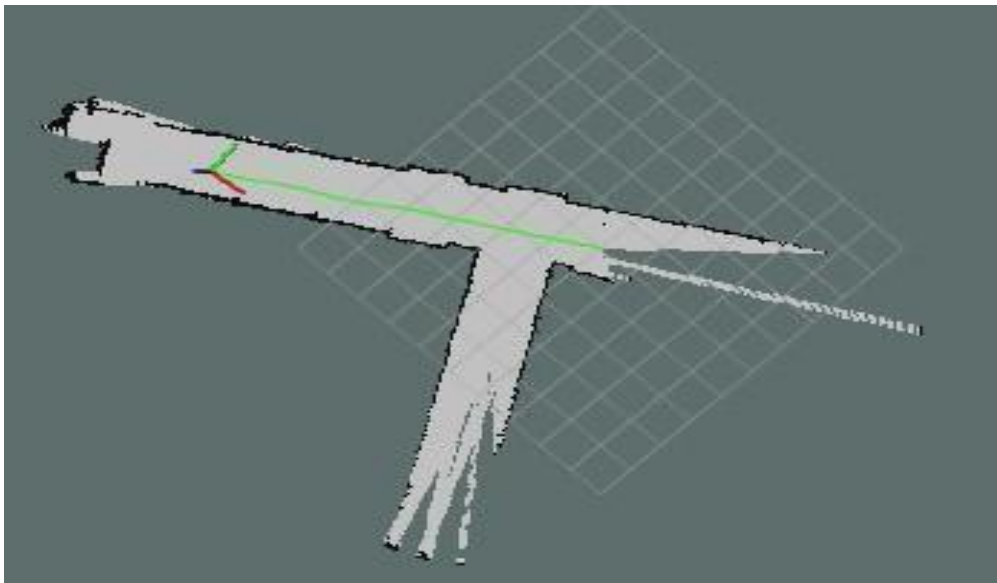


Fig. 5. “HPCC 1st floor Corridors 2D environment map (top-down view)”

It should be noted that the above map is currently is in initial stages and is not fully completed and the final map will be of the entire 1st floor of the HPCC building.

3.7 Conclusion

This chapter explored the selection and implementation of SLAM algorithm for the Pure Zone Rover. Initial challenges with the SLAM Toolbox led us to successfully adopt Hector SLAM, proving effective in mapping dynamic environments. Hector Slam's grid-based mapping and advanced features were validated through testing on our target environments. Looking ahead, our focus will be on refining maps and expanding coverage to the entire 1st of the HPCC building. This chapter represents a significant stride in development of our Pure Zone Rover.

Chapter 4

Autonomous Navigation using Stacks

4.1 Introduction

In this chapter, we explore the technical details of how our self-cleaning robot moves around on its own. We'll look at the special tools it uses, like Lidar sensors, to understand its surroundings accurately. The chapter also discusses the clever algorithms that help the robot plan its movements, showing how it strategically decides where to go. Additionally, we'll examine the control systems that make sure the robot follows its navigation commands precisely. By breaking down these engineering elements, the chapter reveals the complex system that allows our robot to move autonomously.

4.2 Navigation using NAV2 Stack:

4.2.1 Overview of NAV2 stack:

Nav2 is an advanced successor to ROS Navigation Stack, tailored for mobile and surface robotics. It leverages cutting-edge technologies from Autonomous Vehicles, aiming to enable safe execution of complex tasks for mobile robots in diverse environments. Beyond basic navigation, Nav2 supports intermediary poses, object following, and complete coverage navigation. Trusted globally, it's a production-grade framework adopted by 50+ companies. With a focus on optimization, Nav2 provides a suite of functionalities including perception, planning, control, localization, and more, making it a reliable choice for constructing autonomous systems.

4.2.2 Comprehensive Toolset of NAV2:

The comprehensive toolset provided by Nav2 includes functionalities such as map loading, serving, and storing (Map Server), robot localization on maps (AMCL), path planning around obstacles (Nav2 Planner), path following control (Nav2 Controller), path smoothing for continuity (Nav2 Smoother), conversion of sensor data into cost map representations (Nav2 Cost map 2D), behavior tree-based robot behaviors (Nav2 Behavior Trees and BT Navigator), recovery behavior computation (Nav2 Recoveries), sequential waypoint following (Nav2 Waypoint Follower), server lifecycle management (Nav2 Lifecycle

Manager), custom algorithm and behavior plugins (Nav2 Core), collision monitoring (Collision Monitor), Python3 API for interaction (Simple Commander), and velocity smoothing for dynamic feasibility (Velocity Smoother).

4.3 Testing the NAV2 stack on simulated environment:

We conducted a comprehensive assessment of the NAV2 stack in a simulated environment, utilizing Gazebo—a dedicated simulation software for ROS robots. Figure 6 visually captures the essence of the Gazebo simulation we executed. This simulated environment offers crucial components, including a controllable robot model equipped with a simulated lidar, manipulated through teleop commands. Additionally, the environment is furnished with obstacles that the robot must navigate around, serving as a test for the accuracy of the navigation capabilities during its traversal.

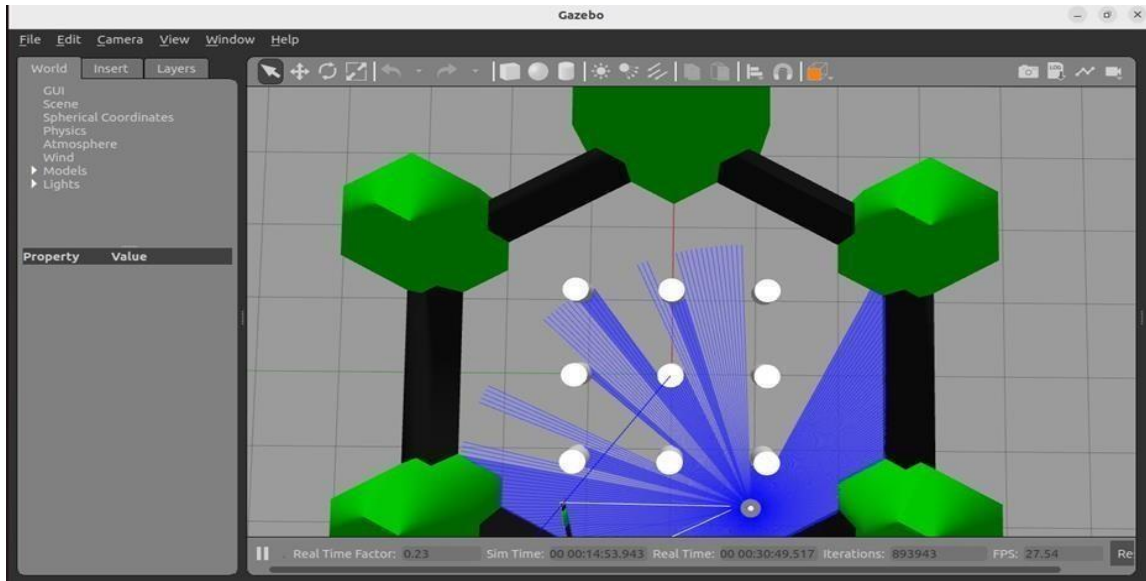


Fig. 6. “Simulated gazebo environment”

4.3.1 Mapping the simulated environment:

Utilizing simulated lidar data generated within the Gazebo environment, we successfully crafted a 2-dimensional map for the simulated world. This task was accomplished through the implementation of the slam toolbox, an essential element of the NAV2 package.

Operating within the Robot Operating System (ROS), the slam toolbox efficiently

subscribes to the scan topic, where simulated lidar data is systematically published by the Gazebo environment. With this data, the slam toolbox adeptly produced a comprehensive map of the simulated Gazebo environment. Furthermore, our customization extended to relocating the lidar, achieved by manipulating the robot model through teleoperation commands. This strategic lidar repositioning enabled us to tailor the map generation process to our specific needs, resulting in the creation of a precise and detailed map, as illustrated in Figure 7.

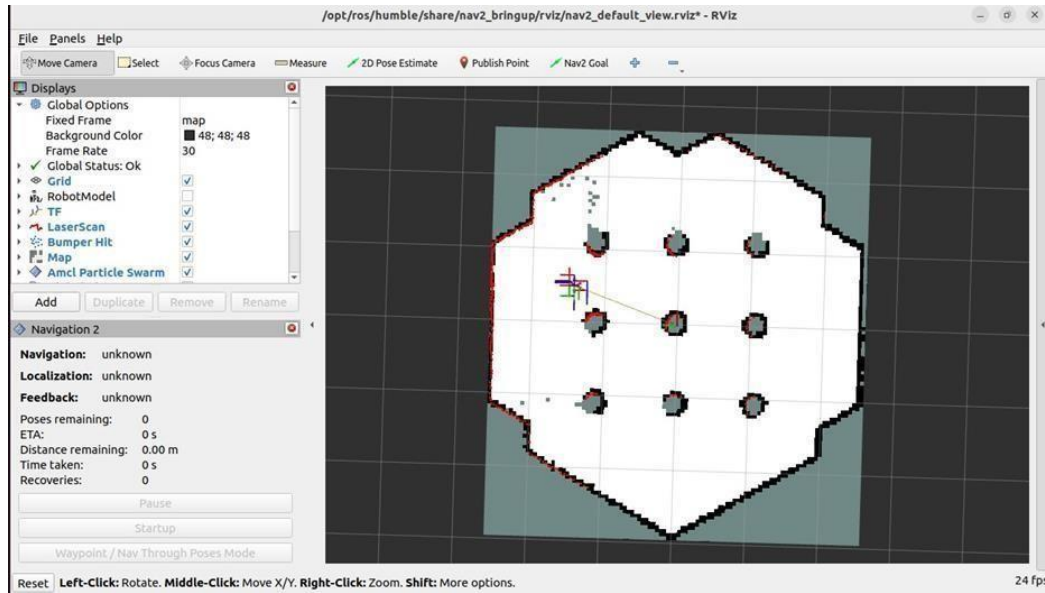


Fig. 7. “Simulated Environment 2D Map (top-down view)”

4.3.2 Real-Time Navigation with Simulated Lidar Map:

Following the map creation, we initiated robot navigation by executing the navigation file within the NAV2 package. Subsequently, we inputted 2D estimation goals into the map, prompting the robot to traverse to the designated positions. As we assigned multiple goals and closely monitored the robot's movements, it became evident that the accuracy of the NAV2 stack was unparalleled, and the algorithm demonstrated exceptional performance. The navigation process can be observed by referring to Figure 8.

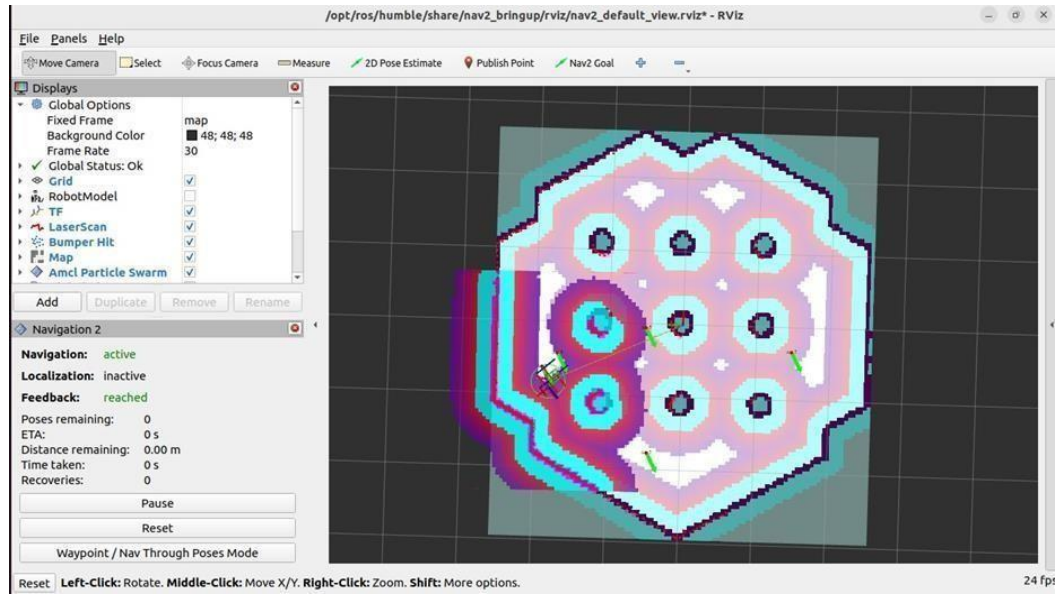


Fig. 8. “Process of navigating the simulated robot”

4.3.3 Complications Faced:

While the NAV2 stack demonstrated flawless performance in a simulated environment and delivered the expected outputs, it's crucial to highlight that, challenges arose when transitioning to real-time implementation on our RP LIDAR. Several compatibility issues emerged, accompanied by suboptimal performance, particularly on a low-processing-power platform. The Raspberry Pi 4, chosen to execute the entire algorithm, revealed limitations in handling the NAV2 stack efficiently due to its constrained processing capabilities.

Faced with these challenges and after thorough consideration, we recognized the need for a strategic approach. Acknowledging the Raspberry Pi's limitations, we opted to address the issues by reverting to an earlier version of the stack. Consequently, we made the decision to downgrade to the Navigation1 stack, also known as the ROS navigation stack, to ensure compatibility and optimal performance.

4.4 Navigating with ROS Navigation Stack: Current Developments

Currently, we're actively integrating the ROS navigation stack into our maps, specifically focusing on the HPPC first-floor map. This targeted effort aims to enhance navigational capabilities within this environment, optimizing the stack's performance to align seamlessly with the intricacies of our designated map for a more tailored navigation experience.

4.5 Conclusion

In conclusion, Chapter 4 delves into the intricate details of autonomous navigation using stacks, particularly focusing on the implementation of the ROS Navigation Stack and the advanced NAV2 stack. The comprehensive toolset offered by NAV2, detailed testing in a simulated environment, and real-time navigation with simulated lidar maps have been explored, showcasing the versatility and reliability of these navigation frameworks. The successful mapping of the simulated environment using lidar data and the subsequent real-time navigation demonstrated the NAV2 stack's exceptional accuracy and performance.

However, challenges arose when transitioning to real-time implementation on the Raspberry Pi, prompting a strategic decision to downgrade to the Navigation1 stack to address compatibility issues and optimize performance. As the team actively integrates the ROS navigation stack into current developments, with a specific focus on the HPPC first-floor map, the goal remains to enhance navigational capabilities and ensure seamless alignment with the intricacies of the designated map. This chapter sheds light on the journey of navigating a self-cleaning robot autonomously, providing valuable insights into the tools, algorithms, and challenges encountered during the implementation process.

Chapter 5

Integrated Systems: Motor Control, Drivers and Structural Foundations

5.1 Introduction

This chapter introduces the pivotal integration of the L293 motor driver with DC motors, seamlessly orchestrated through a Raspberry Pi as the central control unit, and complemented by a meticulously designed power supply. The bidirectional control capabilities of the L293 motor driver are harnessed for precise motor manipulation, while the computational prowess of the Raspberry Pi drives intelligent navigation algorithms essential for our autonomous cleaning robot in this Final Year Project (FYP). Emphasizing the cohesive synergy among these components, the subsequent sections delve into the technical intricacies of motor driver integration, Raspberry Pi interfacing, and crucial power supply considerations. The collaborative integration explored herein ensures the effective execution of cleaning tasks, positioning our robotic system for optimal performance and functionality in the realm of automated cleaning.

5.2 Motor Driver Selection and Overview:

This section delves into the intricacies of selecting the L293 motor driver as a crucial component for our autonomous cleaning robot. We begin by scrutinizing the specific requirements of our DC motors, including voltage and current specifications, and evaluate how well the L293 aligns with these criteria. The dual H-bridge architecture of the L293 is explored in detail, emphasizing its capability for bidirectional motor control, which is fundamental for precise navigation in cleaning tasks. Technical considerations such as thermal performance and compatibility with the Raspberry Pi's GPIO pins are thoroughly examined, highlighting the comprehensive evaluation undertaken to ensure the L293's optimal functionality within our robotic system. This in-depth analysis sets the stage for the subsequent discussion on the seamless integration of the selected motor driver with the Raspberry Pi and the overall success of our autonomous cleaning robot.

5.3 Raspberry Pi Integration:

This section offers a detailed exploration of the integration between the L293 motor driver and the Raspberry Pi, elucidating the significance of employing this single-board computer in our autonomous cleaning robot project. The Raspberry Pi serves as the central nervous system, leveraging its computational power and GPIO pins for precise motor control. We delve into the rationale behind choosing the Raspberry Pi, highlighting its affordability, compact size, and versatile capabilities, which include running sophisticated algorithms for intelligent navigation. The section emphasizes the strategic use of specific GPIO pins for interfacing with the L293 motor driver, outlining the role of each pin in the overall control scheme. By employing the Raspberry Pi, our robotic system benefits from a scalable and programmable platform that facilitates seamless communication and coordination between the motor driver and the cleaning robot's control logic. Refer to figure 9.

Furthermore, the Raspberry Pi integration brings a layer of adaptability and versatility to our autonomous cleaning robot. Its open-source nature allows for the implementation of custom software tailored to our specific needs. We explore the programming aspects involved in translating high-level commands into precise motor control signals, delving into the software architecture that facilitates real-time decision-making for the robot's movements.

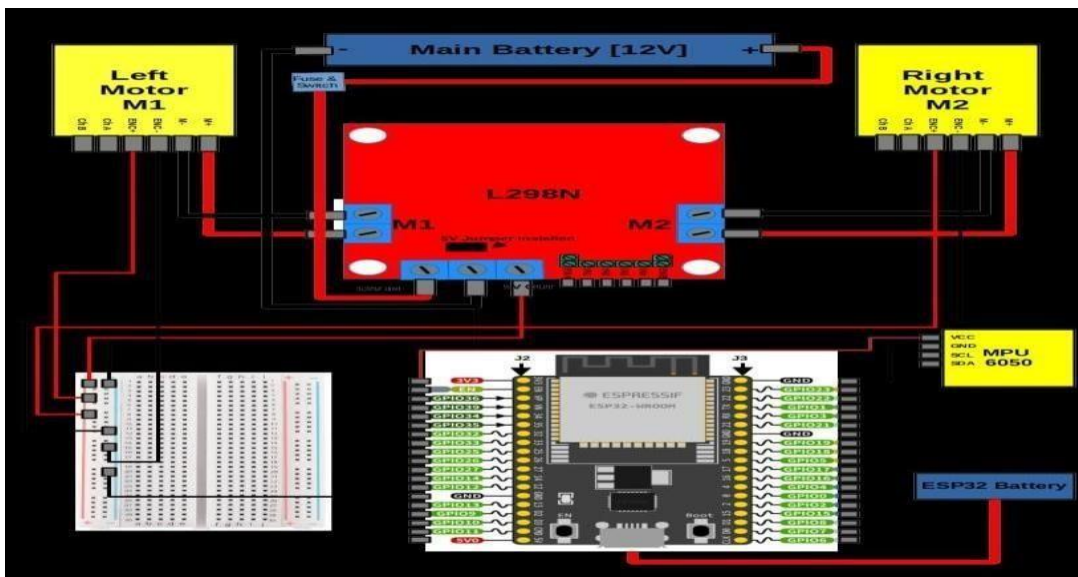


Fig.9. “Connections for Manual Movement of Robot using WIFI”

We are controlling the motor driver using the ESP32 module, which is driving four motors. Although the motor driver is typically used for two motors, we have connected two motors in parallel to each output, allowing us to control four motors simultaneously. The ESP32 receives signals from a web server page, enabling us to control the motors via a mobile device using Wi-Fi. The purpose of this setup was to initially move the robot manually so that it can later operate in both manual and autonomous modes. Currently, we are using manual movement through the ESP32 for mapping purposes.

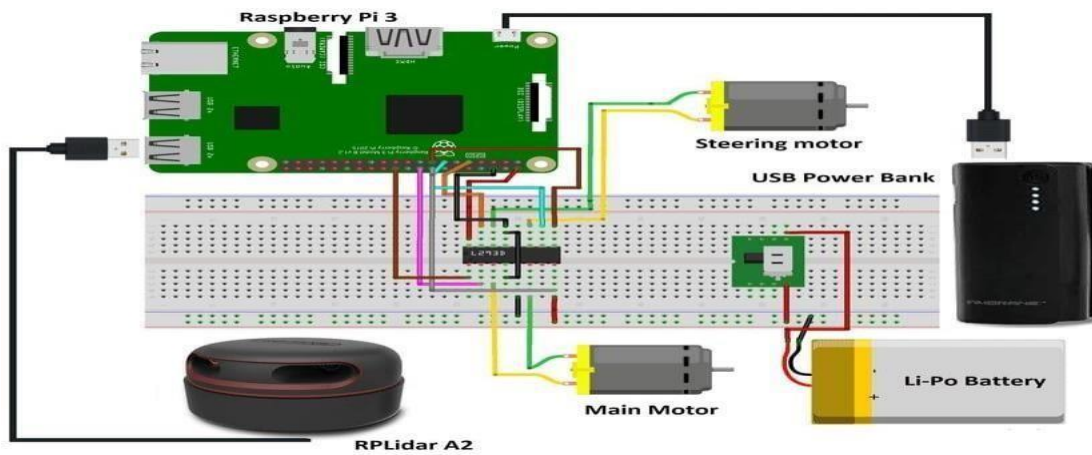


Fig.10. "Connections for Autonomous Movement"

We are currently in the final stages of mapping for the autonomous robot project, with the process almost complete. Following this, our next objective is to initiate the navigation process. While we have made significant progress in mapping, the integration of Raspberry Pi's movement control has not been fully implemented yet.

5.4 Hardware Connections:

Within this section, we meticulously detail the physical connections underpinning the integration of the L293 motor driver with the DC motors and a Lidar sensor in our autonomous cleaning robot. Through comprehensive diagram, we illustrate the wiring layout, emphasizing the specific pins utilized on both the L293 motor driver and the Raspberry Pi, while incorporating connections with the Lidar sensor. We delve into the intricacies of connecting each motor output to the corresponding DC motor, elucidating the

bidirectional control mechanisms of the L293. Furthermore, the power supply connections to the motor driver are expounded upon, ensuring a clear depiction of the energy distribution in the system. On the Raspberry Pi side, we highlight the GPIO pins responsible for interfacing with the motor driver and Lidar sensor, establishing a direct link between computational processes, motor actuation, and environmental sensing. This section serves as an indispensable guide for replicating the precise hardware connections essential for the seamless integration of the motor driver, Lidar sensor, and DC motors in our autonomous cleaning robot. The basic structure/foundation of the rover can be seen in figure 10. It should be noted that this is only the initial structure in which we conducted our initial tests and the final structure of the rover will be quite different.

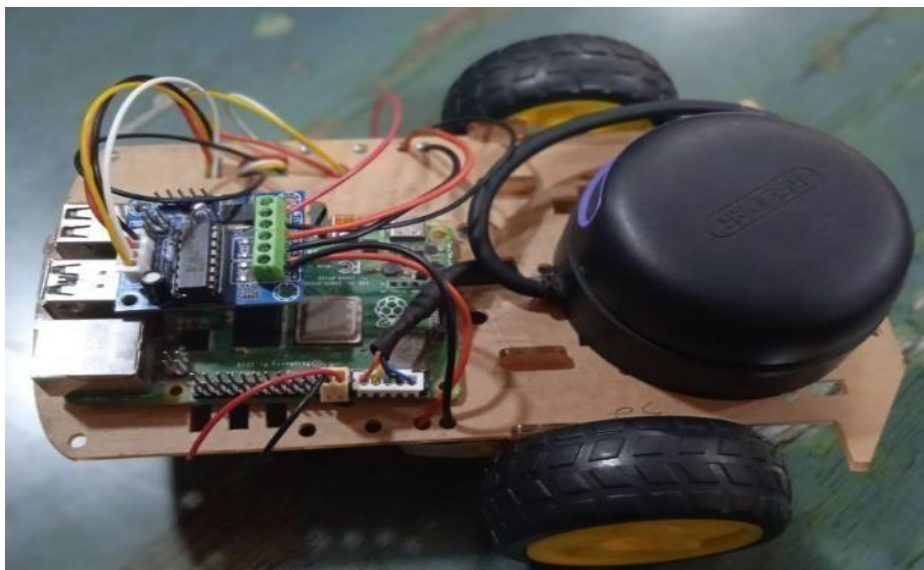


Fig. 11. “Pure Zone Rover base structure (BETA)”

5.5 3D Model Using Blender App:

We created the 3D model of the autonomous robot using Blender app, ensuring accuracy and detail in its design. The model captures the sleek and functional aspects of the robot, highlighting its UV-C lamps, Raspberry components, and LIDAR sensor. The design below at different angles showcases the robot's ability to navigate and sanitize efficiently in various environments, contributing to its overall effectiveness in reducing bacteria and virus transmission.

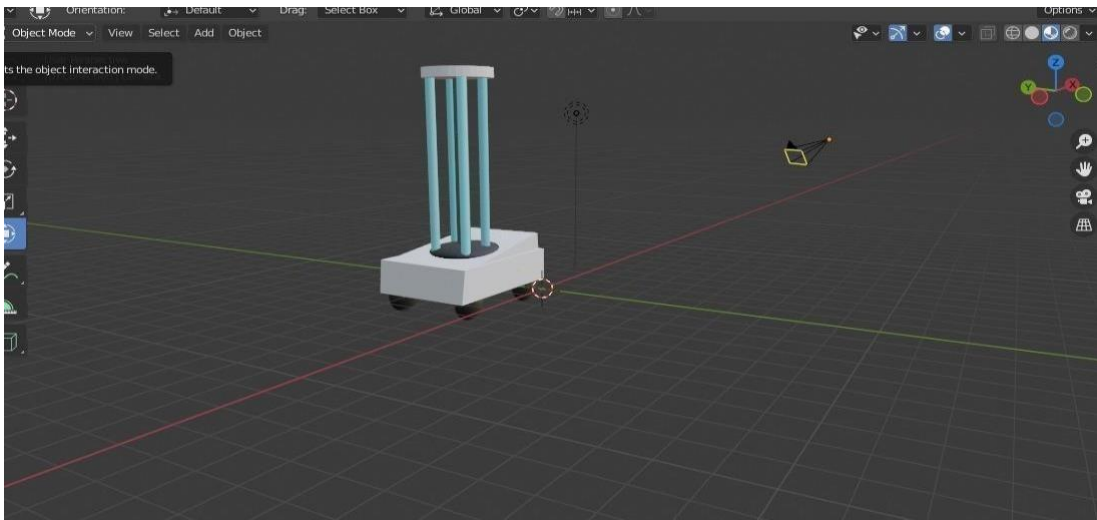


Fig.12. “3D Model of Pure Zone Rover”

5.6 Testing and Calibration:

This section meticulously details the extensive testing and calibration procedures essential for validating the efficacy of our integrated motor control system. We outline the specific metrics used to assess responsiveness and accuracy, describing in-depth methodologies for systematic testing under various conditions. Code snippets utilized in testing are discussed, providing insight into how algorithms translate into real-world motor control actions. Calibration procedures, addressing variations in motor response, environmental factors, and Lidar sensor feedback, are explained. Performance metrics, including motor speed and accuracy of navigation, are presented, offering quantifiable assessments. Challenges encountered are candidly discussed, along with innovative solutions. This comprehensive overview not only validates system robustness but also provides crucial insights for future enhancements in our autonomous cleaning robot.

5.7 Code Explanation:(refer to appendix for the code)

5.7.1 Initialization and Setup

- The code includes the Servo library and declares variables for linear and angular velocities.
- Pin numbers for controlling two motors are defined.
- A servo object is created, and initial position variables are set.
- The `setup` function initializes serial communication, sets motor control pins as outputs, stops the motors initially, and attaches the servo to a specific pin.

5.7.2 Main Loop and Serial Communication

- The `loop` function checks for data available on the serial port.
- If the start character `<` is detected, it reads and parses linear and angular velocities.
- It logs the received values and controls the motors based on these values.
- The servo motor is controlled to move back and forth between set positions, with a delay added to prevent flooding the serial port.

5.7.3 Motor Control Logic

- The `controlMotors` function determines motor speeds and directions based on linear and angular velocities.
- It uses `setMotor1` and `setMotor2` functions to apply these speeds and directions.
- `setMotor1` and `setMotor2` functions set the speed and direction of Motor 1 and Motor 2 using PWM signals.
- `stopMotor1` and `stopMotor2` functions stop the motors by setting PWM values to 0.

5.7.4 Logging and Utility Functions

- The `logReceivedValues` function logs the received linear and angular velocities to the serial monitor, helping in debugging and monitoring the received commands.

5.8. Conclusion:

In summary, the successful integration of the L293 motor driver with DC motors and its smooth collaboration with the Raspberry Pi represent a significant milestone in our autonomous cleaning robot project. The precise documentation of hardware connections, illustrated through circuit diagrams and corresponding code, establishes a sturdy groundwork for replication. Thorough testing and calibration procedures validate the system's responsiveness and versatility, ensuring its dependability across varying conditions. As we transition to ensuing chapters, the focus will shift towards intricate software algorithms governing intelligent navigation, accompanied by a deeper exploration of power supply considerations. This comprehensive approach underscores our dedication to advancing the capabilities of autonomous robotics, specifically in the practical realm of an efficient cleaning robot.

Chapter 6

Enhanced Disinfection: Fog Module and UV Lights Integration

6.1 Introduction

Effective disinfection methods are crucial for maintaining clean environments, especially in high-traffic areas like hospitals and public spaces. Traditional methods, while effective, can be labor-intensive and may not reach all surfaces. Ultraviolet (UV) light and fogging systems offer innovative solutions. UV light damages the DNA or RNA of microorganisms, preventing them from replicating. Fogging systems create a fine mist of disinfectant that reaches inaccessible areas. Combining UV light with fogging enhances disinfection by covering more surface area. This chapter explores the integration of these technologies for improved disinfection outcomes.

6.2 Fog Disinfection Module: Design and Operation

In our fog disinfection module, we use an ultrasonic mist maker submerged in a water reservoir to generate fine droplets of water, creating fog with an optional disinfectant mix. This technology ensures uniform distribution for better surface coverage. The water reservoir serves as a medium for fog generation and holds the disinfectant, allowing for flexibility in choosing disinfectants based on specific needs. This design adapts to different environments, ensuring effective cleaning tailored to each space. Overall, the integration of the ultrasonic mist maker enhances the cleaning process, leading to improved outcomes and a safer environment.



Fig.13. “Ultrasonic Mist Maker”

Referencing from [20]

6.3 UV Lights: Types and Mechanisms of Disinfection

UV lights are classified into three types: UV-A, UV-B, and UV-C. UV-A and UV-B are unsuitable for disinfection and are used in tanning beds and medical phototherapy, respectively. UV-C, however, is germicidal and effectively kills bacteria, viruses, and fungi by damaging their DNA or RNA, preventing replication and causing infections. With a wavelength of around 254 nanometers, UV-C is particularly effective. UV disinfection is a chemical-free method used in water treatment, air purification, and surface disinfection. In our system, UV lights are integrated with a servo motor to enable rotation, ensuring comprehensive coverage, including areas missed by static UV lights.



Fig. 14. “The UV lamp disinfection Robot”

Referencing from [16]

6.4 Integration of Fog Module and UV Lights

The integration of fog disinfection modules with UV lights enhances cleaning efficacy by combining their benefits. UV lights target airborne pathogens and surfaces not reached by fog, while fogging neutralizes pathogens on surfaces. This dual approach ensures thorough sanitation. Additionally, UV lights add flexibility to the disinfection process, allowing for the use of different disinfectant solutions based on specific needs. Overall, this integration offers a comprehensive and effective disinfection solution for various settings.



Fig.15. “Pure Zone Rover structure with UV Lights and Fog Disinfection Module”

6.5 Efficacy and Benefits of Combined Disinfection

The integration of fog disinfection modules with UV lights represents a significant advancement in disinfection technology, offering a range of benefits and enhanced efficacy. This combination ensures comprehensive coverage of surfaces, including hard-to-reach areas, by utilizing the fogging system to neutralize pathogens on surfaces and UV lights to target airborne pathogens. This dual approach not only increases the overall efficacy of the disinfection process but also reduces the reliance on chemical disinfectants, leading to a safer and more environmentally friendly disinfection method. Additionally, the integrated system allows for shorter disinfection times compared to traditional methods, making it more efficient and convenient for various applications. Overall, the combination of fog disinfection modules and UV lights provides a highly effective and efficient solution for achieving thorough disinfection in diverse environments.

6.6 Safety Measures and Considerations

The use of fog disinfection modules and UV lights for disinfection requires careful safety measures. UV light can be harmful to the skin and eyes, so protective gear like goggles and gloves is essential. Exposure should be limited to prevent adverse effects. Proper handling and ventilation are crucial for fog disinfection modules to minimize exposure to disinfectant solutions. Regular equipment maintenance is also necessary for effectiveness.

and safety. Operators should receive proper training and education on risks and safety precautions. Additionally, environmental impact should be minimized through proper waste disposal. These measures ensure that the use of fog disinfection modules and UV lights is effective and safe.

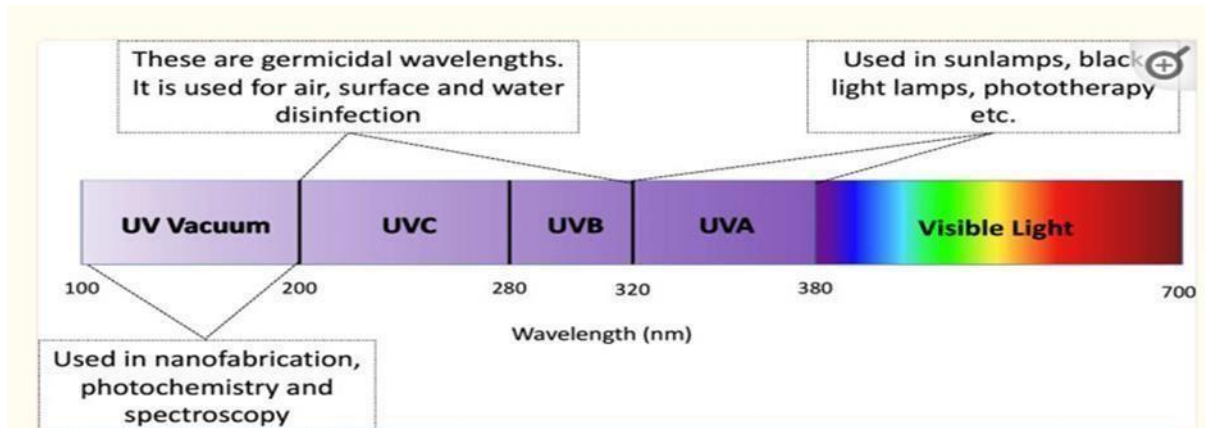


Fig. 16.

The ultraviolet spectrum and its applications: UVC and UVB light (200–320 nm) are known to have germicidal capabilities. UVA radiation (320–400 nm) is not germicidal, and vacuum UV (100–200 nm) is absorbed in air rapidly and is not used for surface disinfection

Referencing from [19]

6.7 Conclusion

The integration of fog disinfection modules with UV lights represents a significant advancement in disinfection technology, offering a highly effective and efficient solution for achieving thorough sanitation in various environments. By combining these technologies, comprehensive coverage of surfaces is ensured, including hard-to-reach areas, leading to enhanced cleaning efficacy. Furthermore, the use of UV lights and fogging systems reduces the reliance on chemical disinfectants, making the disinfection process safer and more environmentally friendly. The integration of UV lights with servo motors adds a dynamic element to the disinfection process, allowing for a more thorough sanitization of the environment. Overall, the combination of fog disinfection modules, UV lights, and servo motors provides a comprehensive and effective solution for achieving thorough disinfection. This integrated approach not only ensures the safety of individuals and the environment but also offers a more efficient and convenient method for maintaining clean spaces.

Chapter 7

Transitioning to Final Phase - System Architecture and Core Functionalities

This chapter marks the transition from our mid-year progress to the final phase of developing our Pure-zone rover. It delves into the refined system architecture and core functionalities that are pivotal for our rover's advanced navigation system. We'll explore how Hector SLAM is used to construct detailed and accurate maps of the environment and how the ROS navigation stack leverages these maps to guide the rover's movements. Additionally, we will cover the installation and configuration of ROS topics, their role in communication, and the significance of the transform tree in maintaining spatial relationships and ensuring precise navigation.

7.1 Introduction

As we progress from the mid-year phase of our project to its final stages, it is crucial to focus on the sophisticated aspects of our rover's navigation system. This chapter builds upon our initial work by addressing the system's architecture and the advanced functionalities necessary for effective autonomous operation. The emphasis is on refining the components and integrating them seamlessly to achieve reliable navigation, obstacle avoidance, and targeted disinfection.

7.2 ROS Topic Installation

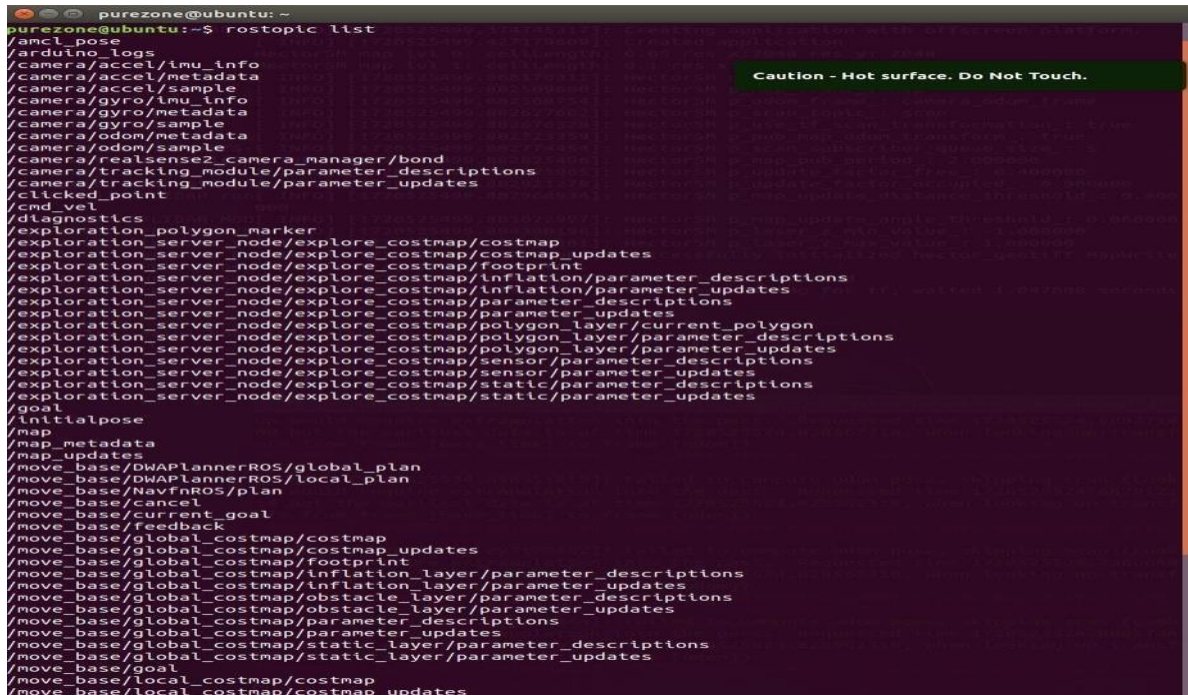
With the transition to the final phase, the installation and configuration of ROS topics have been updated to ensure robust communication between nodes. These topics are essential for the accurate flow of data between sensors and control systems.

7.2.1 Topic Installation and Configuration

We have meticulously configured communication channels for sensor data, control commands, and status updates. The key topics include:

- **Laser Scan Data (/scan):** Provides comprehensive 360-degree laser scans from the RP Lidar.
- **Odometry Data (/odom):** Delivers real-time position and velocity data from the tracking device.

- **Velocity Commands (/cmd_vel):** Sends linear and angular velocity commands to the motor drivers.

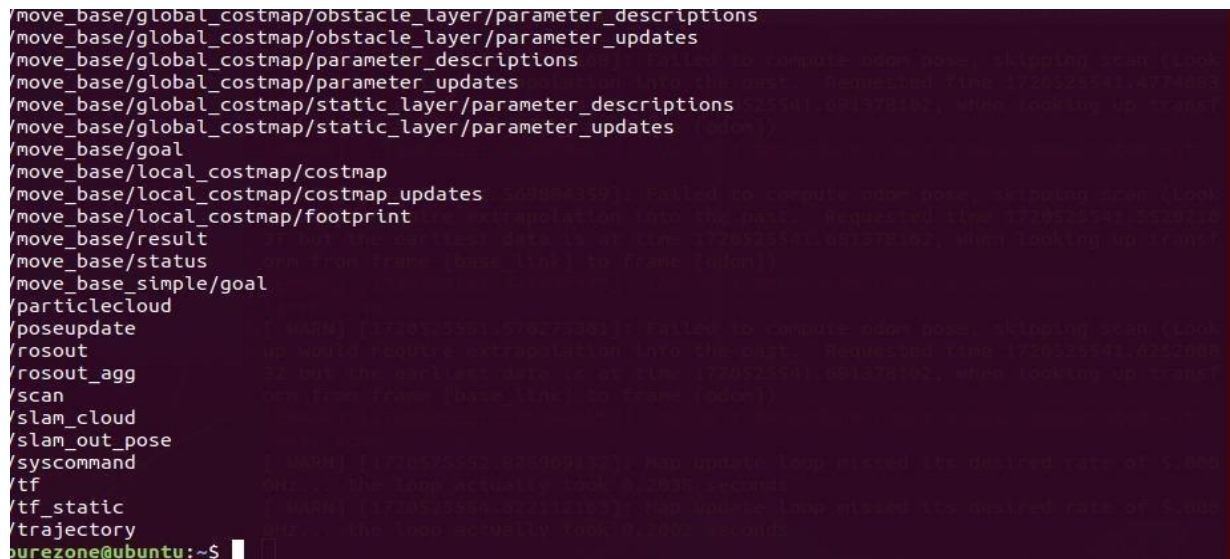


```

purezone@ubuntu:~$ rostopic list
/amcl_pose
/arduino_logs
/camera/accel/imu_info
/camera/accel/metadata
/camera/accel/sample
/camera/gyro/imu_info
/camera/gyro/metadata
/camera/gyro/sample
/camera/odom/metadata
/camera/odom/sample
/camera/realSense2_camera_manager/bond
/camera/tracking_module/parameter_descriptions
/camera/tracking_module/parameter_updates
/clicked_point
/cmd_vel
/diagnostics
/exploration_server_node/explore_costmap/costmap
/exploration_server_node/explore_costmap/costmap_updates
/exploration_server_node/explore_costmap/footprint
/exploration_server_node/explore_costmap/inflation/parameter_descriptions
/exploration_server_node/explore_costmap/inflation/parameter_updates
/exploration_server_node/explore_costmap/parameter_descriptions
/exploration_server_node/explore_costmap/parameter_updates
/exploration_server_node/explore_costmap/polygon_layer/current_polygon
/exploration_server_node/explore_costmap/polygon_layer/parameter_descriptions
/exploration_server_node/explore_costmap/polygon_layer/parameter_updates
/exploration_server_node/explore_costmap/sensor/parameter_descriptions
/exploration_server_node/explore_costmap/sensor/parameter_updates
/exploration_server_node/explore_costmap/static/parameter_descriptions
/exploration_server_node/explore_costmap/static/parameter_updates
/goal
/initialpose
/map
/map_metadata
/map_updates
/move_base/DWAPlannerROS/global_plan
/move_base/NavfnROS/plan
/move_base/cancel
/move_base/current_goal
/move_base/feedback
/move_base/global_costmap/costmap
/move_base/global_costmap/costmap_updates
/move_base/global_costmap/footprint
/move_base/global_costmap/inflation_layer/parameter_descriptions
/move_base/global_costmap/inflation_layer/parameter_updates
/move_base/global_costmap/obstacle_layer/parameter_descriptions
/move_base/global_costmap/obstacle_layer/parameter_updates
/move_base/global_costmap/parameter_descriptions
/move_base/global_costmap/parameter_updates
/move_base/global_costmap/static_layer/parameter_descriptions
/move_base/global_costmap/static_layer/parameter_updates
/move_base/goal
/move_base/local_costmap/costmap
/move_base/local_costmap/costmap_updates

```

Fig 17. (a)



```

/move_base/global_costmap/obstacle_layer/parameter_descriptions
/move_base/global_costmap/obstacle_layer/parameter_updates
/move_base/global_costmap/parameter_descriptions
/move_base/global_costmap/parameter_updates
/move_base/global_costmap/static_layer/parameter_descriptions
/move_base/global_costmap/static_layer/parameter_updates
/move_base/goal
/move_base/local_costmap/costmap
/move_base/local_costmap/costmap_updates
/move_base/local_costmap/footprint
/move_base/result
/move_base/status
/move_base_simple/goal
/particlecloud
/poseupdate
/rosout
/rosout_agg
/scan
/sl原因am_cloud
/sl原因am_out_pose
/syscommand
/tf
/tf_static
/trajectory
purezone@ubuntu:~$

```

Fig17. (b) “ROS Topic List”

7.3 Hector SLAM: Building a Map of the World

Hector SLAM is a robust Simultaneous Localization and Mapping (SLAM) algorithm that plays a critical role in our system. SLAM algorithms address a fundamental challenge in robotics: building a map of the environment while simultaneously determining the robot's location within that map. Hector SLAM achieves this by leveraging the data acquired from our RP Lidar sensor.

```
/home/purezone/catkin_ws/src/hector_slam/hector_slam_launch/launch/tutorial.launch http://localhost:11311
ROS_MASTER_URI=http://localhost:11311

process[hector_mapping-1]: started with pid [31730]
process[map_to_camera_odom_broadcaster-2]: started with pid [31731]
process[base_to_camera_broadcaster-3]: started with pid [31735]
process[base_laser_broadcaster-4]: started with pid [31745]
process[base_to_odom_broadcaster-5]: started with pid [31766]
process[hector_trajectory_server-6]: started with pid [31785]
process[hector_geotiff_node-7]: started with pid [31792]
[ INFO] [1720525178.204465453]: Creating application with offscreen platform.
[ INFO] [1720525178.295795803]: Created application
[ INFO] [1720525178.372471714]: Waiting for tf transform data between frames /map and scanmatcher_frame to become
available
HectorSM map lvl 0: cellLength: 0.05 res x:2048 res y: 2048
HectorSM map lvl 1: cellLength: 0.1 res x:1024 res y: 1024
[ INFO] [1720525178.823969660]: HectorSM p_base_frame_: camera_odom_frame
[ INFO] [1720525178.847164223]: HectorSM p_map_frame_: map
[ INFO] [1720525178.847310003]: HectorSM p_odom_frame_: camera_odom_frame
[ INFO] [1720525178.847422085]: HectorSM p_scan_topic_: scan
[ INFO] [1720525178.847535521]: HectorSM p_use_tf_scan_transformation_: true
[ INFO] [1720525178.847597083]: HectorSM p_pub_map_odom_transform_: true
[ INFO] [1720525178.847704061]: HectorSM p_scan_subscriber_queue_size_: 5
[ INFO] [1720525178.847771561]: HectorSM p_map_pub_period_: 2.000000
[ INFO] [1720525178.847837133]: HectorSM p_update_factor_free_: 0.400000
[ INFO] [1720525178.848094734]: HectorSM p_update_factor_occupied_: 0.900000
[ INFO] [1720525178.848242493]: HectorSM p_map_update_distance_threshold_: 0.400000
[ INFO] [1720525178.848339107]: HectorSM p_map_update_angle_threshold_: 0.060000
[ INFO] [1720525178.848395513]: HectorSM p_laser_z_min_value_: -1.000000
[ INFO] [1720525178.848484418]: HectorSM p_laser_z_max_value_: 1.000000
[ INFO] [1720525178.869903270]: Successfully initialized hector_geotiff MapWriter plugin TrajectoryMapWriter.
[ INFO] [1720525178.870112799]: Geotiff node started
[ INFO] [1720525179.390437700]: Finished waiting for tf, waited 1.018048 seconds
```

Fig 18. "Hector SLAM Package in ROS"

7.3.1 Lidar Scans and Point Cloud Data

The RP Lidar (A2M7 model) is a Light Detection and Ranging (LiDAR) sensor. It emits pulsed laser beams and measures the reflected light to determine the distance to objects in its surroundings. This process generates precise 360-degree laser scans of the environment. Hector SLAM utilizes these scans to build a map. Each scan essentially represents a "slice" of the environment from the Lidar's perspective.



Fig. 19. "RP-Lidar A2M7 model"

Referencing from [21]

7.3.2 Building the Map: From Scans to Representation

Hector SLAM employs a technique called "scan matching" to construct the map. It takes two or more laser scans and aligns them, accounting for the robot's movement between scans. This alignment process allows Hector SLAM to identify common features in the environment and gradually build a cohesive map representation. Over time, as the robot navigates, Hector SLAM incorporates additional scans, refining and expanding the map.

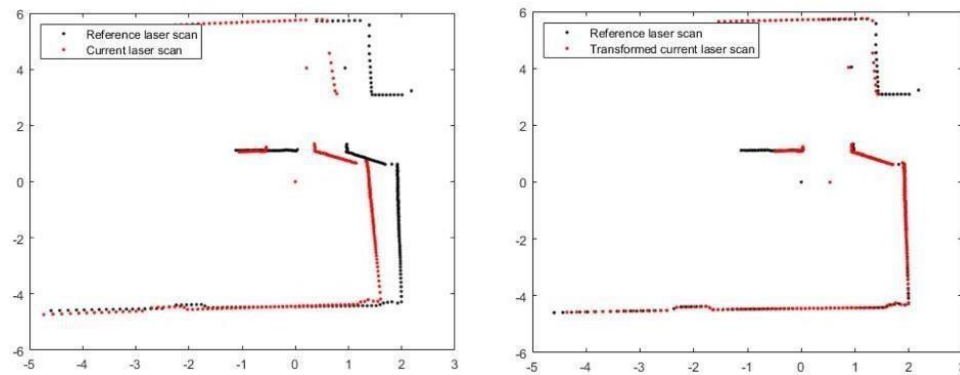


Fig 20. Scan Matching between two laser scans

7.3.3 Addressing Challenges: Noise and Uncertainty

Real-world sensor data is inherently noisy, and robot movement can be imprecise. Hector SLAM incorporates techniques to handle these uncertainties. It employs statistical methods to account for potential errors in sensor readings and robot odometry (position estimation based on movement). This ensures the generated map is as accurate and reliable as possible.

7.4 ROS Navigation Stack

The ROS navigation stack is a collection of software tools and algorithms designed for robot navigation within a mapped environment. It utilizes the map generated by Hector SLAM and sensor data to plan and execute safe and efficient movement for the rover. Figure 21. shows the recent ROS navigation framework.

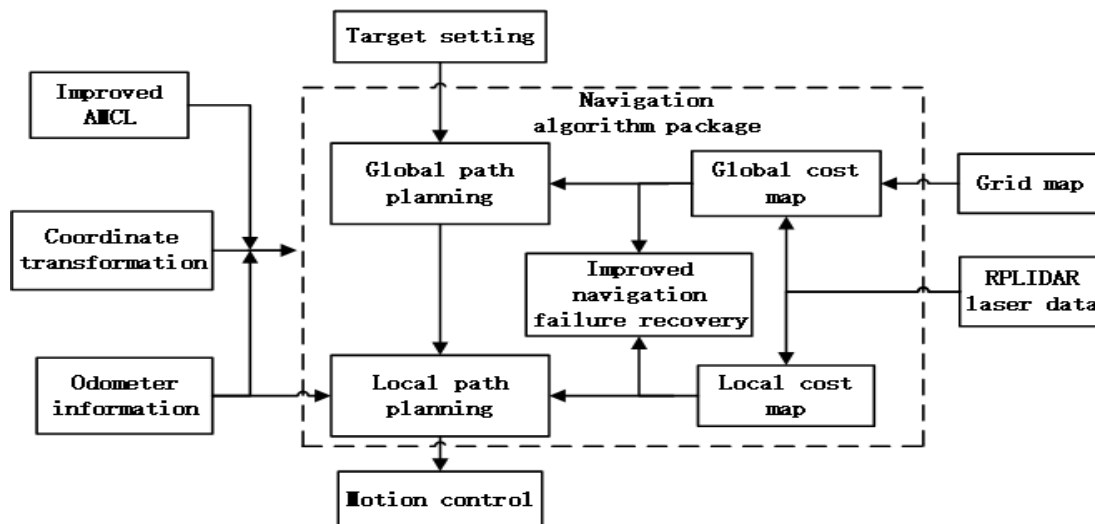


Fig 21. “Ros navigation framework”

7.4.1 The Navigation Stack Workflow

Sensor Data Acquisition: The navigation stack retrieves real-time sensor data from various sources, including:

- **Laser Scans from Lidar:** These scans provide crucial information about obstacles in the robot's path.
- **Odometry Data from Tracking Device:** This data, often obtained from inertial measurement units (IMUs) or visual sensors, continuously updates the robot's position within the map.
- **Map and Transform Integration:** The navigation stack utilizes the map generated by Hector SLAM and incorporates data on the robot's pose (position and orientation) within the map using transforms. Transforms define the relationship between different coordinate frames, ensuring all components have a consistent understanding of the robot's location relative to the environment.
- **Path Planning:** Based on the map, sensor data, and the desired goal location, the navigation stack employs path planning algorithms to determine the optimal trajectory for the robot. These algorithms consider factors like obstacle avoidance, path efficiency, and curvature constraints.
- **Velocity Control:** The navigation stack calculates and publishes linear and angular velocity commands on the `command_velocity` topic. These velocity commands are then sent to the motor drivers, controlling the robot's movement and steering it along the planned path. The navigation stack continuously monitors sensor data and may adjust the velocity commands in real-time to ensure safe and efficient navigation.

7.4.2 Key Components and Integration

Coordinate Frames and Transforms: Defines the spatial relationships between different reference points, including both static and dynamic transforms.

Navigation Integration: Ensures accurate data alignment and pose estimation for effective path planning and control.

7.4.3 Managing Transform Updates

Utilizes the ROS TF library for broadcasting, listening, and visualizing transforms, maintaining up-to-date spatial information and enabling real-time adjustments.

7.5 Transform Tree: Ensuring Accurate Spatial Relationships

The transform tree is essential for managing spatial relationships and integrating data from various components. It provides a unified reference frame that ensures consistent spatial understanding across the system, which is crucial for accurate navigation and operation.

7.5.1 Key Components and Integration

Coordinate Frames and Transforms: Defines the spatial relationships between different reference points, including both static and dynamic transforms.

Navigation Integration: Ensures accurate data alignment and pose estimation for effective path planning and control.

7.5.2 Managing Transform Updates

Utilizes the ROS TF library for broadcasting, listening, and visualizing transforms, maintaining up-to-date spatial information and enabling real-time adjustments. Figure 22 and Figure 23 highlight the “Transform tree” and “RQT graph”.

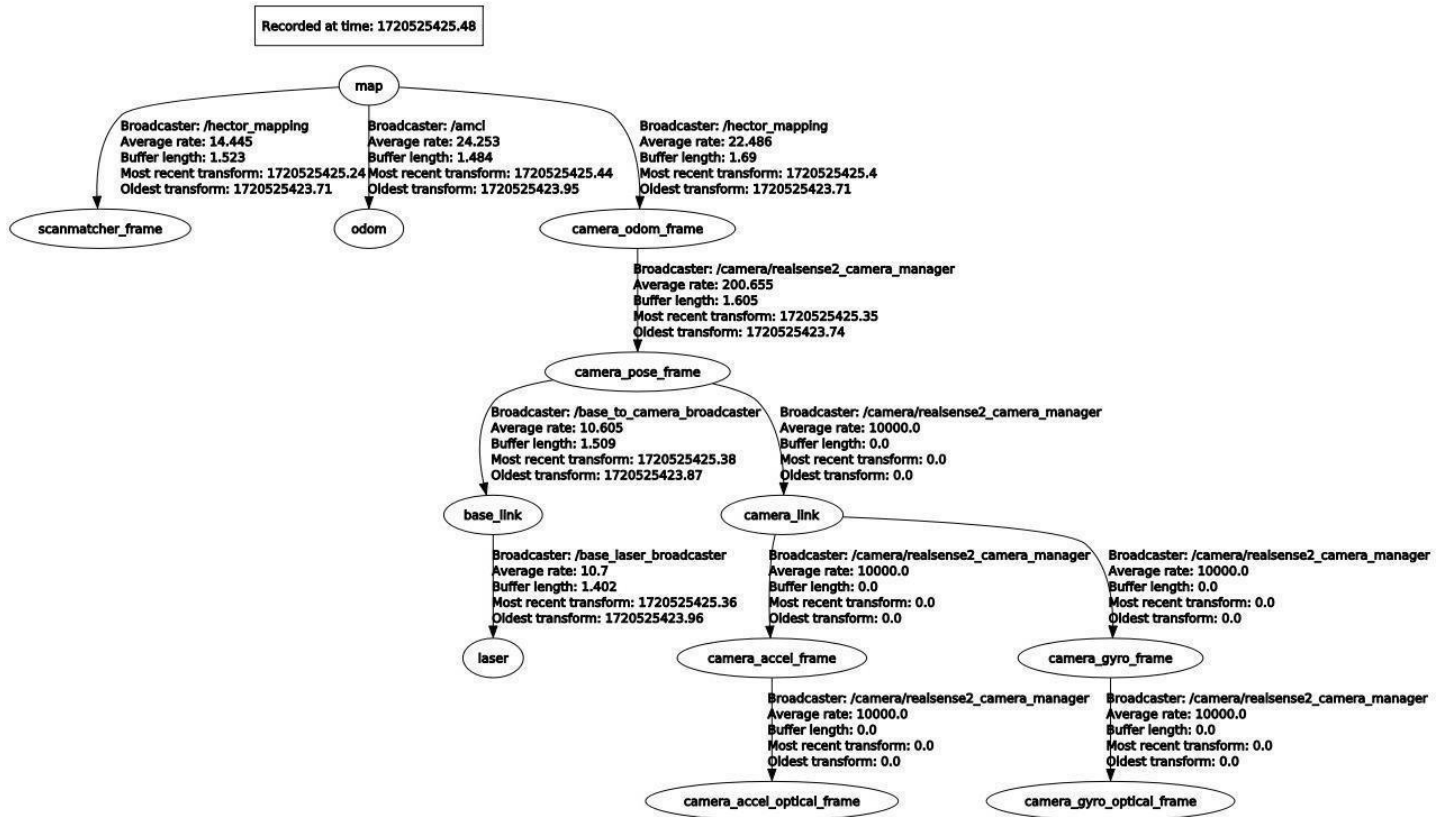


Fig 22. "Transform Tree"

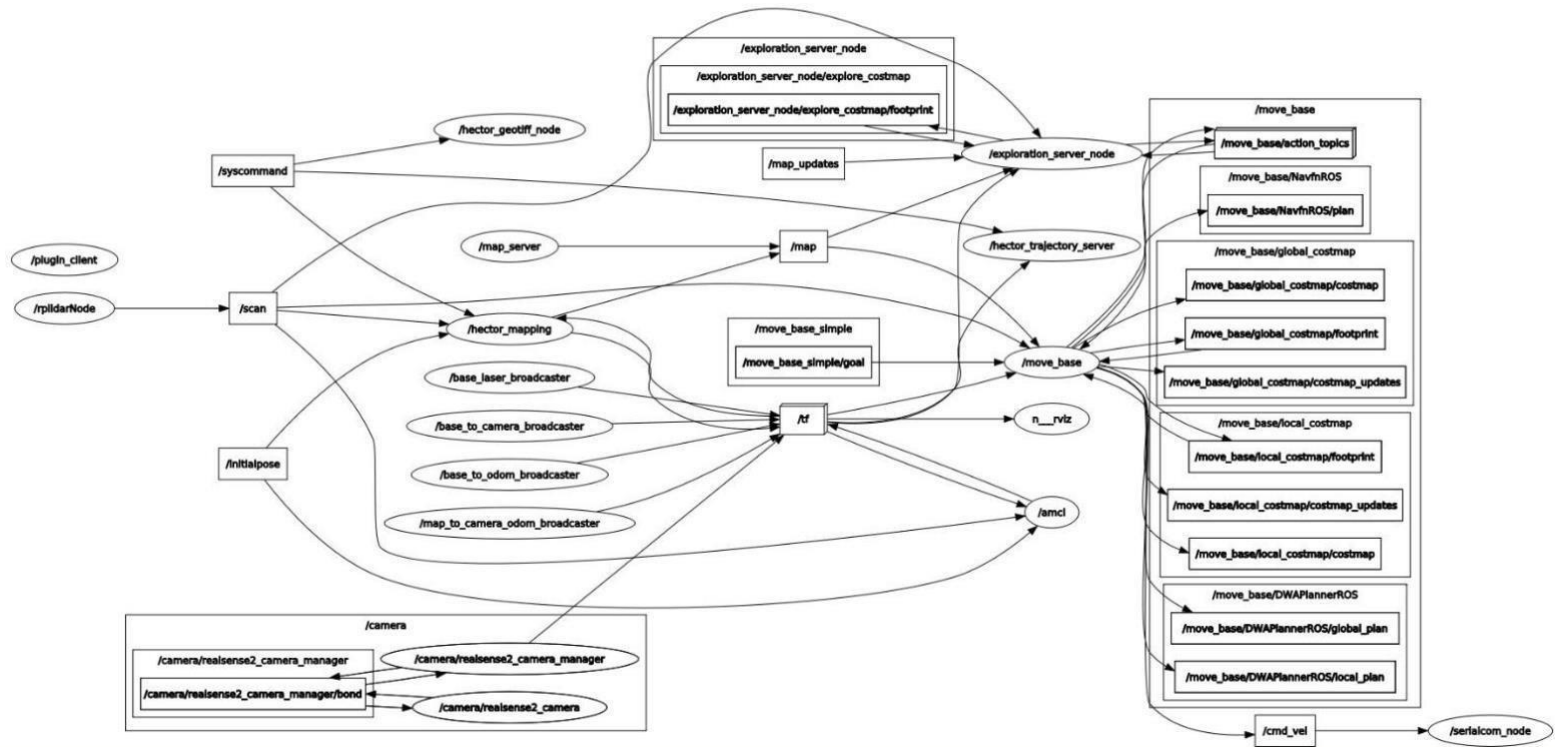


Fig 23. "RQT- Graph"

7.6 Additional Components

In this phase, we have also integrated the Exploration Server and implemented a bash file:

- **Exploration Server:** The Exploration Server is a critical component for managing autonomous exploration tasks. It leverages the maps generated by Hector SLAM to systematically explore unknown areas. The server operates by defining exploration goals and planning routes based on the current environmental map. It ensures that the rover navigates efficiently, avoids obstacles, and optimizes area coverage. By processing real-time sensor data, the Exploration Server makes dynamic decisions to adapt the exploration strategy, thereby enhancing the rover's ability to autonomously cover designated areas and achieve targeted disinfection.

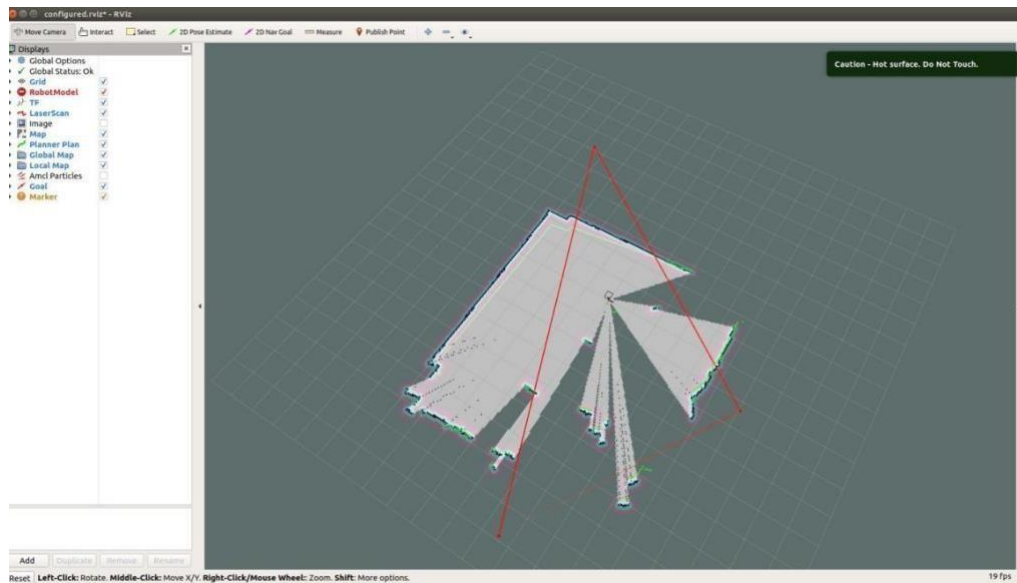


Fig 24. “Exploration server ”

- **Bash File:** The bash file facilitates the setup and execution of system components by automating several tasks. It handles the initialization of ROS nodes, configuration of parameters, and launching of essential processes. This script streamlines the deployment process, reduces manual setup time, and ensures consistency in the execution of software components. By integrating components like Hector SLAM, the ROS navigation stack, and the Exploration Server, the bash file supports a seamless and efficient operational workflow for the rover.

```

purezone@ubuntu:~$ ./sensors.sh
purezone@ubuntu:~$ ./slam.sh

[ INFO ] [1720523966.879394228]: Creating application with offscreen platform.
HectorSM map lvl 0: cellLength: 0.05 res x:2048 res y: 2048
[ INFO ] [1720523967.011398558]: Created application
HectorSM map lvl 1: cellLength: 0.1 res x:1024 res y: 1024
[ INFO ] [1720523967.154366367]: HectorSM p_base_frame_: camera_odom_frame
[ INFO ] [1720523967.177480312]: HectorSM p_map_frame_: map
[ INFO ] [1720523967.17936319]: HectorSM p_odom_frame_: camera_odom_frame
[ INFO ] [1720523967.178451234]: HectorSM p_scan_topic_: scan
[ INFO ] [1720523967.17839582]: HectorSM p_use_tf_scan_transformation_: true
[ INFO ] [1720523967.179154021]: HectorSM p_pub_map_odom_transform_: true
[ INFO ] [1720523967.179292253]: HectorSM p_scan_subscriber_queue_size_: 5
[ INFO ] [1720523967.180133641]: HectorSM p_map_pub_period_: 2.000000
[ INFO ] [1720523967.180774134]: HectorSM p_update_factor_free_: 0.400000
[ INFO ] [1720523967.180897993]: HectorSM p_update_factor_occupied_: 0.900000
[ INFO ] [1720523967.181184410]: HectorSM p_map_update_distance_threshold_: 0.400
000
[ INFO ] [1720523967.181366386]: HectorSM p_map_update_angle_threshold_: 0.060000
[ INFO ] [1720523967.181458066]: HectorSM p_laser_z_min_value_: -1.000000
[ INFO ] [1720523967.183005156]: HectorSM p_laser_z_max_value_: 1.000000
[ INFO ] [1720523967.825514077]: Successfully initialized hector_geotiff MapWrite
r plugin TrajectoryMapWriter.
[ INFO ] [1720523967.826524845]: Geotiff node started
[ INFO ] [1720523967.841315038]: Finished waiting for tf, waited 1.021904 seconds

NODES
/
  exploration_server_node (exploration_server/exploration_server_node)
  plugin_client (exploration_server/plugin_client)
ROS_MASTER_URI=http://localhost:11311
process[plugin_client-1]: started with pid [9646]
process[exploration_server_node-2]: started with pid [9648]
[ INFO ] [1720523971.698589576]: Please use the 'Point' tool in Rviz to select an
exploration boundary.
[ WARN ] [1720523971.798655807]: Change marker topic to exploration_polygon_marke
r before continuing.
[ INFO ] [1720523972.630326742]: explore_costmap: Using plugin "static"
[ INFO ] [1720523972.860464666]: Requesting the map...
[ INFO ] [1720523973.144990224]: Resizing costmap to 2048 X 2048 at 0.050000 n/pl
x
[ INFO ] [1720523973.288223207]: Received a 2048 X 2048 map at 0.050000 n/plx
[ INFO ] [1720523973.288468164]: Subscribing to updates
[ INFO ] [1720523973.462990021]: explore_costmap: Using plugin "polygon_layer"
[ INFO ] [1720523973.754666790]: explore_costmap: Using plugin "sensor"
[ INFO ] [1720523973.824992212]: Subscribed to Topics: laser
[ INFO ] [1720523974.315245983]: explore_costmap: Using plugin "inflation"

[ WARN ] [1720523979.641384576]: Failed to compute odom pose, skipping scan (Look
up would require extrapolation into the past. Requested time 1720523979.6413845
76 but the earliest data is at time 1720523979.839111881, when looking up transf
orm from frame [base_link] to frame [odom])
[ERROR] [1720523989.78432719]: Couldn't determine robot's pose associated with
laser scan
[ WARN ] [1720523989.784314962]: Failed to compute odom pose, skipping scan (Look
up would require extrapolation into the past. Requested time 1720523979.7078629
86 but the earliest data is at time 1720523979.839111881, when looking up transf
orm from frame [base_link] to frame [odom])
[ERROR] [1720523989.784300379]: Couldn't determine robot's pose associated with
laser scan
[ WARN ] [1720523989.784542470]: Failed to compute odom pose, skipping scan (Look
up would require extrapolation into the past. Requested time 1720523979.7865383
70 but the earliest data is at time 1720523979.839111881, when looking up transf
orm from frame [base_link] to frame [odom])
[ERROR] [1720523989.784520494]: Couldn't determine robot's pose associated with
laser scan
[ WARN ] [1720523989.784779042]: Failed to compute odom pose, skipping scan (Look
up would require extrapolation into the past. Requested time 1720523979.8541513
02 but the earliest data is at time 1720523979.913276150, when looking up transf
orm from frame [base_link] to frame [odom])
[ERROR] [1720523989.784873160]: Couldn't determine robot's pose associated with
laser scan

```

Fig 25. “ROS Bash File”

7.7 Conclusion

In conclusion, this chapter highlights the transition to the final phase of our autonomous rover's development, focusing on the integration of Hector SLAM and the ROS navigation stack. These enhancements ensure precise mapping, efficient path planning, and robust communication through updated ROS topics and the transform tree. This comprehensive approach solidifies the foundation for effective and accurate operation, achieving the goals set for the final stage of our project.

Chapter 8: System Integration [Hardware implementation]

8.1 Introduction

This chapter dives deep into the heart of our autonomous UV and fog disinfection rover – the integration of various hardware and software components. We'll explore the reasoning behind our shift from Raspberry Pi to Jetson Nano, delve into the intricacies of integrating tracking devices for odometry, and unpack the implementation of RP Lidar for efficient mapping and navigation.

8.2 Challenges with Raspberry Pi

Our initial foray involved running algorithms on a Raspberry Pi. While it's a popular choice for hobbyist projects, we encountered limitations that hindered its effectiveness for our more demanding application:

- **Limited Processing Power:** The Raspberry Pi's processing capabilities proved insufficient to handle the combined demands of the power-hungry RP Lidar and tracking device running simultaneously. This resulted in performance bottlenecks, causing data loss or sluggish operation when both sensors were active.
- **Terminal-Only Operation:** A Cumbersome Interface: Operating solely through the command line interface (CLI) on the Raspberry Pi proved cumbersome and inefficient. Launching, monitoring, and debugging multiple software packages became a time-consuming hassle, hindering development progress.
- **Node Management Constraints:** Scaling Limitations: Due to the Raspberry Pi's limited processing power, it struggled to manage the increased number of nodes required after implementing SLAM (Simultaneous Localization and Mapping). SLAM algorithms inherently involve multiple software components working together, and the Raspberry Pi simply couldn't handle the computational load of these additional nodes.



Fig 26. “ Jetson nano vs Raspberry Pi ”

Referencing From [22]

8.3 Transition to Jetson Nano

To overcome these limitations and ensure smooth, reliable operation, we transitioned to the Nvidia Jetson Nano. This powerful single-board computer offered significant advantages.

8.3.1 Overview of Nvidia Jetson Nano

The Nvidia Jetson Nano is a small, powerful computer designed specifically for AI and machine learning applications. It is part of Nvidia's Jetson family of products, which are used in various edge computing and robotics projects. The Jetson Nano is well-suited for projects that require real-time processing, such as autonomous robots, drones, and smart devices.

8.3.2 Key Specifications

- **CPU:** Quad-core ARM Cortex-A57 MPCore processor, which provides robust processing capabilities for multitasking and handling complex computations.
- **GPU:** 128-core NVIDIA Maxwell architecture GPU, designed to handle parallel processing tasks efficiently, making it ideal for AI, computer vision, and deep learning applications.
- **Memory:** 4GB of LPDDR4 memory, which ensures smooth performance even when running memory-intensive applications.
- **Storage:** Supports microSD card storage, allowing for easy expansion and sufficient storage for operating systems, applications, and data.
- **Connectivity:** Includes multiple USB 3.0 and USB 2.0 ports, a Gigabit Ethernet port, MIPI CSI-2 camera interface, HDMI and DisplayPort outputs, and GPIO pins. This extensive connectivity supports various peripherals and sensors, enhancing the versatility of the Jetson Nano.
- **Enhanced Graphical User Interface (GUI):** The Jetson Nano boasts a user-friendly GUI, providing a significant improvement over the Raspberry Pi's terminal-only environment. This GUI streamlines the execution of software packages, simplifies system monitoring, and facilitates debugging efforts. Developers can easily launch and manage multiple applications simultaneously, significantly improving development efficiency.



Fig 27. "GUI of Jetson Nano"

- **Superior Computational Power:** The Jetson Nano packs a significant processing power punch compared to the Raspberry Pi. This allows for seamless operation with both the tracking device and RP Lidar running concurrently. The increased processing power ensures efficient data handling and smooth execution of complex algorithms like SLAM and navigation, enabling our rover to operate effectively in real-time.

- **Advanced GPU Capabilities:**

The Nvidia Jetson Nano's 128-core GPU is designed to accelerate parallel processing tasks, making it ideal for computer vision and AI applications. It can handle tasks such as object detection, image classification, and neural network inference efficiently, allowing our rover to make real-time decisions based on sensor data.

- **Robust Connectivity Options**

The Jetson Nano offers a wide array of connectivity options, including:

USB Ports: Four USB 3.0 ports and one USB 2.0 port for connecting peripherals such as cameras, sensors, and external storage.

Ethernet: Gigabit Ethernet port for fast and reliable network connections, essential for remote monitoring and control.

GPIO Pins: 40-pin GPIO header for connecting additional sensors and modules.

Camera Interface: MIPI CSI-2 interface for high-quality camera input.

Improved Software Support Nvidia's JetPack SDK, specifically designed for the Jetson

platform, provides comprehensive support for AI, machine learning, and robotics applications. The SDK includes libraries and tools such as CUDA, cuDNN, and TensorRT, which are optimized for the Jetson Nano, facilitating the development and deployment of sophisticated algorithms.

- **Energy Efficiency:**

Despite its powerful performance, the Jetson Nano is designed to be energy-efficient. It supports power modes that allow us to optimize power consumption based on the task requirements. This feature is particularly beneficial for our rover, as it ensures prolonged operation without frequent battery replacements or recharges, making the system more reliable and cost-effective.

- **Versatile Development Environment**

The Jetson Nano supports multiple development environments, including popular frameworks like TensorFlow, PyTorch, and OpenCV. This versatility enables our team to leverage existing knowledge and resources, speeding up the development process. The compatibility with various programming languages such as Python, C++, and Java further broadens our development capabilities.

8.4 Integration of Sensors and Software: Building the Core

8.4.1 RP Lidar Integration (A2M7 Model)

The RP Lidar (A2M7 model) serves as the eye of our rover, providing critical environmental data. It integrates seamlessly using the rplidar ROS package. This sensor functions as a Light Detection and Ranging (LiDAR) device, emitting pulsed laser beams and measuring the reflected light to determine the distance to objects in its surroundings. The RP Lidar provides precise 360-degree laser scans of the environment, creating a rich point cloud data set. These scans are the foundation for Hector SLAM, enabling the creation of detailed maps for navigation.


```
/home/purezone/catkin_ws/src/rplidar_ros/launch/rplidar_a2m7.launch http://localhost:113
* /rplidarNode/serial_port: /dev/ttyUSB0

NODES
/
  rplidarNode (rplidar_ros/rplidarNode)

auto-starting new master
process[master]: started with pid [8672]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to f95e734e-3de4-11ef-a958-0242164c8727
process[rosout-1]: started with pid [8767]
started core service [/rosout]
process[rplidarNode-2]: started with pid [8770]
[ INFO] [1720523915.881326343]: RPLIDAR running on ROS package rplidar_ros, SDK
Version:2.1.0
[ INFO] [1720523915.893423553]: RPLIDAR MODE:A2M7
[ INFO] [1720523915.893525275]: RPLIDAR S/N: 59FDEDF9C7E29BD1A7E39EF2C92E431B
[ INFO] [1720523915.893576267]: Firmware Ver: 1.32
[ INFO] [1720523915.893690438]: Hardware Rev: 6
[ INFO] [1720523915.894547500]: RPLidar health status : OK.
[ INFO] [1720523916.173176938]: current scan mode: Sensitivity, sample rate: 16
Khz, max_distance: 16.0 m, scan frequency:10.0 Hz,
```

Fig 28. “RP Lidar Package”

8.4.2 Tracking Device Integration (Realsense Tracking Module): Keeping Track of the Rover's Position

For accurate and reliable navigation, we utilize a tracking device – the Realsense tracking module. This device integrates through its dedicated ROS package and provides real-time odometry data. Odometry refers to the process of estimating a robot's position and orientation based on its movement. The precise coordinates obtained from the tracking device, often using visual or inertial sensors, are essential for the navigation algorithm to determine the robot's location within the mapped environment. This real-time position data allows the rover to navigate precisely and avoid obstacles.



*Fig 29. “Realsense tracking
module” Referencing from
[23]*

```
/home/purezone/catkin_ws/src/realsense-ros/realsense2_camera/launch/rs_t265.launch http://localhost:11311
* /camera/realsense2_camera/wait_for_device_timeout: -1.0
* /rostdistro: melodic
* /rosversion: 1.14.13

NODES
  /camera/
    realsense2_camera (nodelet/nodelet)
    realsense2_camera_manager (nodelet/nodelet)

ROS_MASTER_URI=http://localhost:11311

process[camera/realsense2_camera_manager-1]: started with pid [27408]
process[camera/realsense2_camera-2]: started with pid [27409]
INFO [1720524844.309387300]: Initializing nodelet with 4 worker threads.
INFO [1720524844.741261032]: RealSense ROS v2.3.2
INFO [1720524844.741350619]: Built with LibRealSense v2.50.0
INFO [1720524844.741453903]: Running with LibRealSense v2.50.0
INFO [1720524844.860981601]:
INFO [1720524844.895316768]: Device with serial number 119622110535 was found.
INFO [1720524844.895689853]: Device with physical ID 2-1.1-3 was found.
INFO [1720524844.895962101]: Device with name Intel RealSense T265 was found.
INFO [1720524844.897405429]: Device with port number 2-1.1 was found.
INFO [1720524844.897686896]: Device USB type: 3.1
INFO [1720524844.919004769]: No calib_odom_file. No input odometry accepted.
INFO [1720524844.919736199]: getParameters...
INFO [1720524845.300131977]: setupDevice...
INFO [1720524845.300224532]: JSON file is not provided
INFO [1720524845.300281826]: ROS Node Namespace: camera
INFO [1720524845.300328390]: Device Name: Intel RealSense T265
INFO [1720524845.300376048]: Device Serial No: 119622110535
INFO [1720524845.300427612]: Device physical port: 2-1.1-3
INFO [1720524845.300490739]: Device FW version: 0.2.0.951
INFO [1720524845.300541105]: Device Product ID: 0x0B37
INFO [1720524845.300590742]: Enable PointCloud: Off
INFO [1720524845.300636004]: Align Depth: Off
INFO [1720524845.300681422]: Sync Mode: Off
INFO [1720524845.300755591]: Device Sensors:
INFO [1720524845.300976900]: Tracking Module was found.
INFO [1720524845.301083883]: (Depth, 0) sensor isn't supported by current device! -- Skipping...
INFO [1720524845.301151958]: (Color, 0) sensor isn't supported by current device! -- Skipping...
INFO [1720524845.301197272]: (Confidence, 0) sensor isn't supported by current device! -- Skipping...
..
INFO [1720524845.301278525]: num_filters: 0
INFO [1720524845.301333370]: Setting Dynamic reconfig parameters.
WARN [1720524845.326531264]: Param '/camera/tracking_module/frames_queue_size' has value 256 that
s not in range [0, 32]. Removing this parameter from dynamic reconfigure options.
INFO [1720524845.371607141]: Done Setting Dynamic reconfig parameters.
INFO [1720524845.371984497]: gyro stream is enabled - fps: 200
INFO [1720524845.372147263]: accel stream is enabled - fps: 62
INFO [1720524845.372504358]: pose stream is enabled - fps: 200
INFO [1720524845.372974269]: setupPublishers...
INFO [1720524845.404608099]: setupStreams...
INFO [1720524845.433410588]: SELECTED BASE: Pose, 0
INFO [1720524845.444099161]: RealSense Node Is Up!
WARN [1720524845.445135080]:
```

Fig 30. “Realsense tracking module Package “

8.5 Software Stack: The Symphony of ROS, Hector SLAM, and NavigationStack

The software foundation for our system is built upon three key components working in concert: Robot Operating System (ROS 1), Hector SLAM, and the ROS navigation stack.

- **ROS 1: The Maestro of Communication:** ROS 1 acts as a meta-operating system specifically designed for robot development. It provides a framework for communication between various software components, enabling seamless data exchange and coordinated operation. ROS facilitates communication using topics (data streams) and services (remote procedure calls), allowing different software nodes (programs) to work together efficiently.

- **Hector SLAM: Building a Map of the World:** This robust SLAM algorithm leverages the laser scans acquired from the RP Lidar. Hector SLAM constructs a detailed map of the environment by aligning and merging multiple scans. This map serves as a crucial reference for the navigation process. Hector SLAM employs advanced techniques to account for sensor noise and robot movements, resulting in a highly accurate and reliable map.

- **Navigation Stack:** The ROS navigation stack comprises a collection of tools and algorithms that facilitate robot navigation within an environment. It utilizes various inputs to guide the rover's movement:
 - **Laser Scans from Lidar:** These scans, obtained from the RP Lidar, are used for obstacle detection and avoidance. The navigation stack analyzes the point cloud data from the scans to identify potential obstacles in the robot's path and plan a safe trajectory.
 - **Transforms: The Language of Coordinates:** Transforms define the relationship between different coordinate frames within the ROS environment. They enable seamless communication between various nodes (software components) within the system. Transforms are crucial for the navigation stack to understand the position and orientation of the robot relative to the map and obstacles.
 - **Odometry Data: Real-Time Position Updates:** The tracking device provides real-time odometry data, continuously updating the robot's position within the map. This data, often incorporating information from inertial measurement units (IMUs) or visual sensors, allows the navigation stack to account for any slippage or drift in the robot's movement, ensuring accurate positioning.

8.6 Navigation Algorithm – Real-Time Control in Action

The navigation algorithm operates in real-time to ensure smooth and efficient robot movement. Here's an in-depth look at its key functions:

- **Concurrent Mapping and Navigation: Building and Using the Map on the Fly:** Hector SLAM continuously builds the map while the navigation algorithm utilizes it for real-time decision-making. This allows the robot to navigate in unknown environments and adapt to changes. The navigation stack can leverage the partially built map from Hector SLAM to plan safe paths even as the map continues to be refined.
- **Position Estimation: Knowing Where We Are:** The navigation algorithm employs map pointers to accurately estimate the robot's position within the map. This estimation is crucial for determining the optimal path for navigation. The navigation stack uses various techniques like particle filters or Kalman filters to account for uncertainties in sensor data and robot movement, providing a reliable estimate of the robot's position.

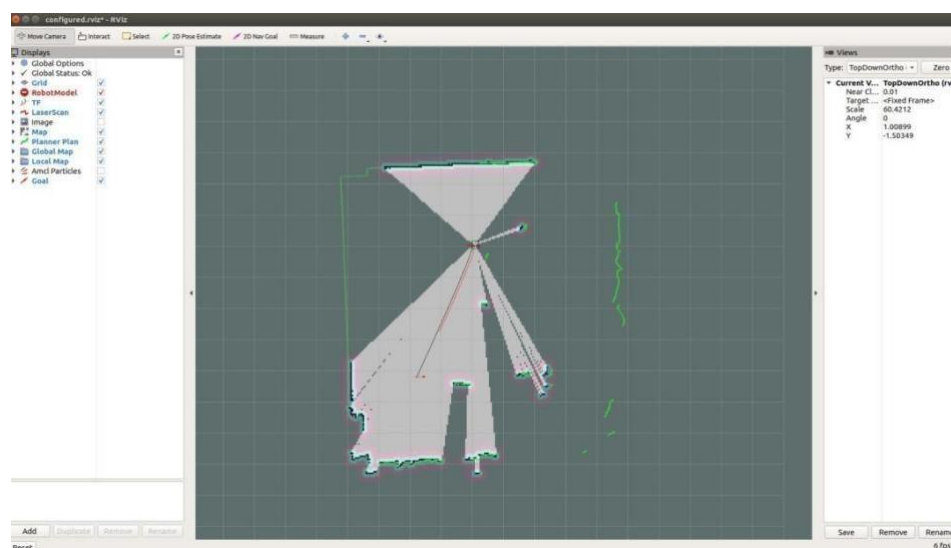


Fig 31.(a)

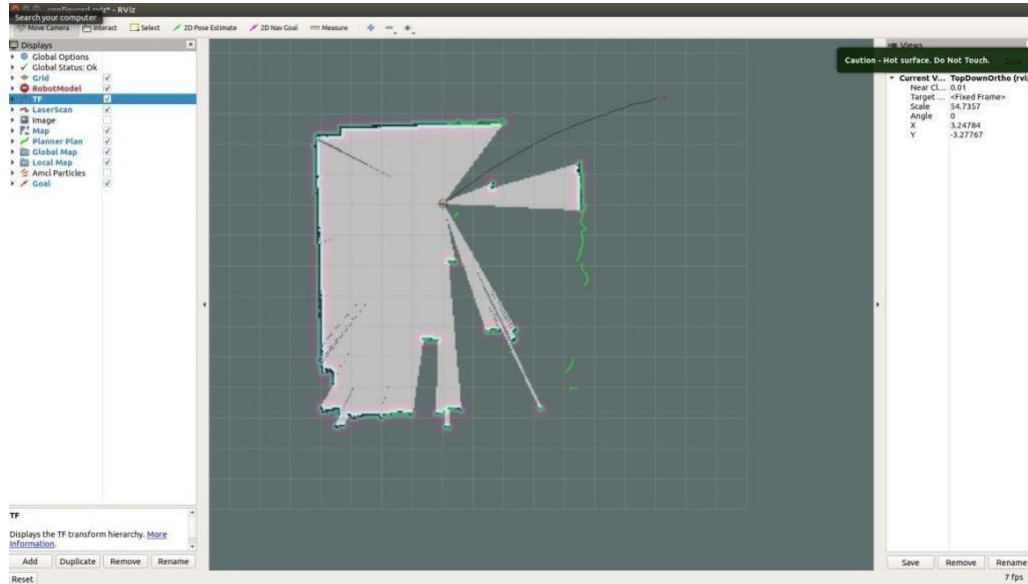


Fig 31. (b)
Fig 31. (a) and (b) representing “Navigation examples”

- **Command Velocity Generation: Steering the Rover:** Based on the map information, sensor data, and position estimation, the algorithm calculates and publishes linear and angular velocity values on the `command_velocity` topic. These velocity commands are then sent to the motor drivers of the rover, controlling its movement and steering it along the planned path. The navigation stack considers factors like obstacle avoidance, desired speed, and path curvature when calculating the appropriate velocity commands.

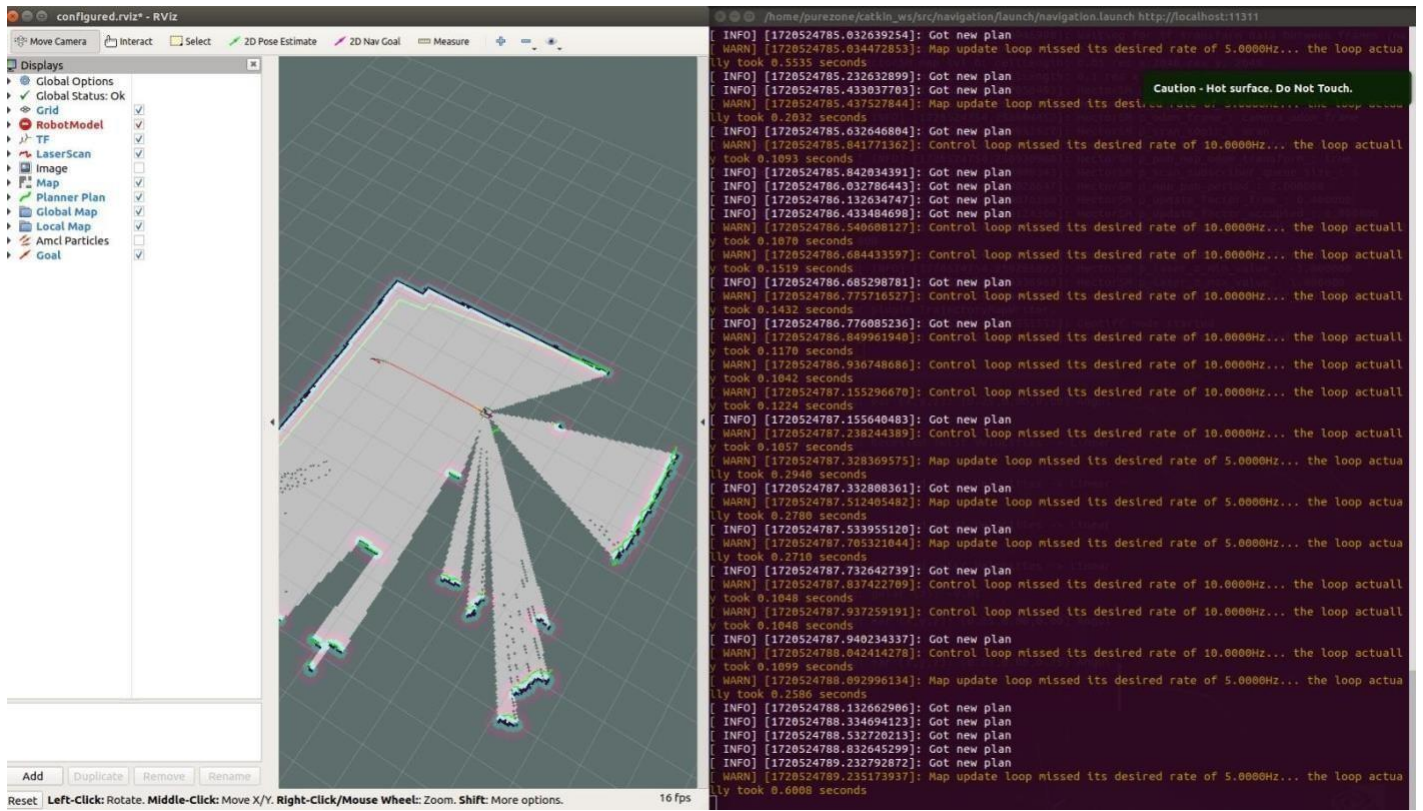


Fig 32. “Command Velocity Package”

8.7 Conclusion

The transition to Jetson Nano and the integration of RP Lidar and the tracking device significantly improved our system's performance. This robust hardware and software platform provides a reliable foundation for autonomous navigation. ROS 1 facilitates seamless communication between various components, Hector SLAM builds accurate maps, and the navigation stack efficiently guides the rover through its environment.

These advancements ensure smooth, real-time operation and pave the way for further development of our Autonomous UV and fog disinfection Rover.

Chapter 9

Implementation Details

9.1 Introduction

This chapter delves into the practical implementation aspects of our autonomous UV and fog disinfection rover. We'll explore how the core functionalities discussed in Chapter 7 and 8 are implemented on the hardware platform and how the UV and fog disinfection systems are integrated.

9.2 Hardware Integration

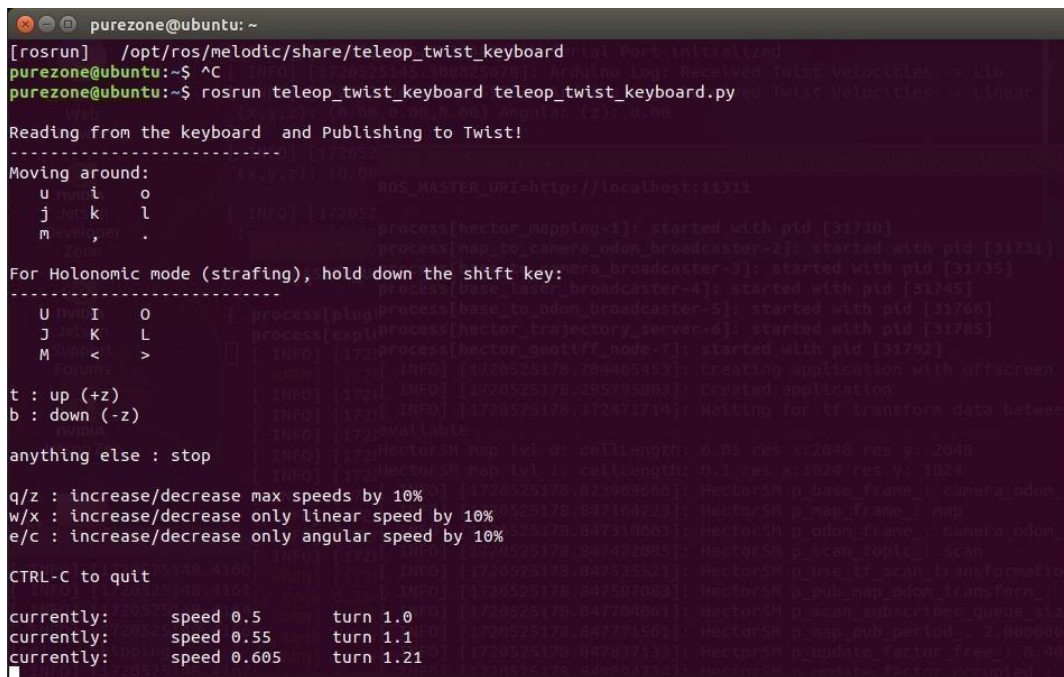
Our rover leverages a combination of powerful hardware components to achieve its autonomous navigation and disinfection goals:

- **Nvidia Jetson Nano:** As the central processing unit (CPU), the Jetson Nano runs the core software functionalities – Hector SLAM and the ROS navigation stack. Its superior processing power enables real-time map building, path planning, and navigation control.
- **RP Lidar (A2M7 Model):** This sensor plays a crucial role in SLAM by providing 360-degree laser scans of the environment. The Jetson Nano communicates with the RP Lidar through the rplidar ROS package for data acquisition.
- **Tracking Device (Realsense Tracking Module):** This device integrates with the Jetson Nano through its dedicated ROS package and provides real-time odometry data. This data is essential for the navigation stack to maintain accurate position estimates within the map.
- **Motor Driver and Arduino Uno:** For motor control, we utilize an Arduino Uno board interfaced with the Jetson Nano. The Jetson Nano communicates with the Arduino Uno, sending high-level velocity commands (linear and angular) via serial communication. The Arduino Uno translates these commands into appropriate control signals for the motor driver, regulating the rover's movement.

9.3 Motor Driver Control: Translating Commands into Movement

The Jetson Nano, running the ROS navigation stack, calculates the desired linear and angular velocities for the rover based on the map, sensor data, and the planned path. These velocity commands are published on the “**cmd_vel topic**”. The Arduino Uno subscribes to this topic and receives the velocity commands from the Jetson Nano. The Arduino Uno then processes these commands and generates the appropriate pulse-width modulation (PWM) signals for the motor driver. By varying the duty cycle of the PWM signal, the Arduino Uno controls the speed and direction of the rover's motors.

Additionally, our system supports **manual** control through the “**teleop_twist_keyboard**” package, allowing operators to manually drive the rover using keyboard commands. This provides flexibility in scenarios where autonomous navigation may not be feasible or desired.



```
[rosrun] /opt/ros/melodic/share/teleop_twist_keyboard (ctrl) port initialized
purezone@ubuntu:~$ ^C
purezone@ubuntu:~$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py
Reading from the keyboard and Publishing to Twist!
-----
Moving around:
  u      i      o
  j      k      l
  m      ,      .

For Holonomic mode (strafing), hold down the shift key:
  U      I      O
  J      K      L
  M      <      >

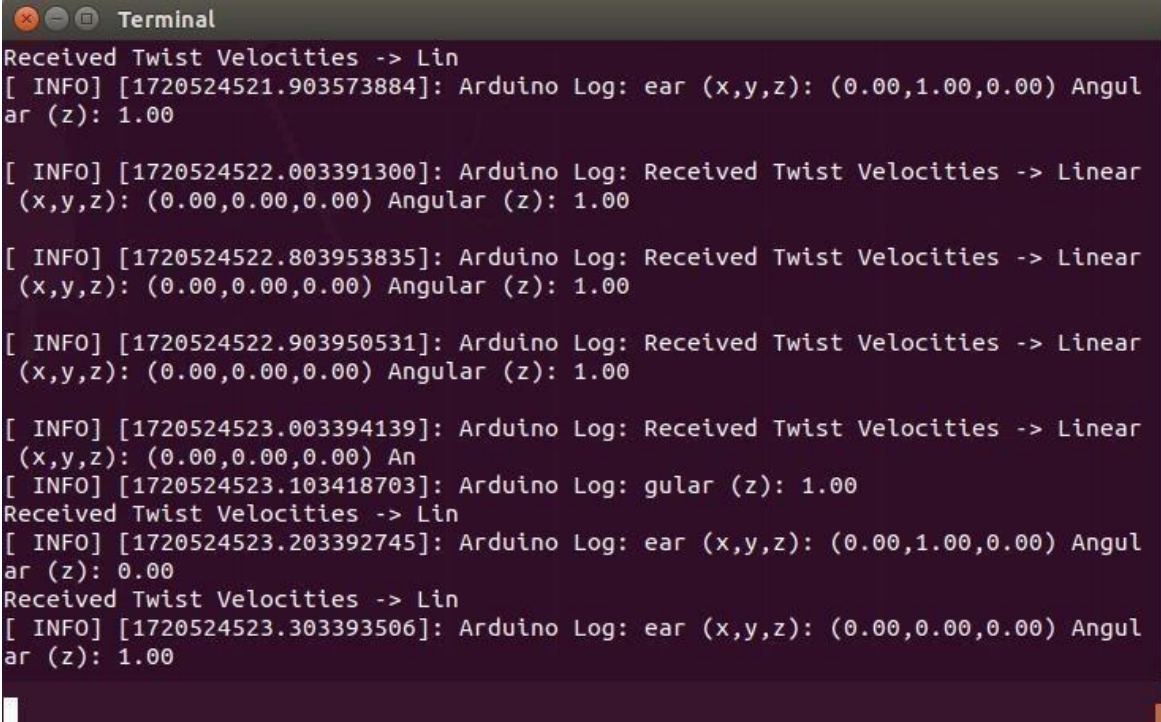
t : up (+z)
b : down (-z)
anything else : stop

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%

CTRL-C to quit

currently: speed 0.5      turn 1.0
currently: speed 0.55     turn 1.1
currently: speed 0.605    turn 1.21
```

Fig 33. (a) “Running Teleop Twist Command on ROS”

A terminal window titled "Terminal" with a dark background and light-colored text. It displays a series of log messages from an Arduino. The messages include timestamps in brackets, followed by "[INFO]" and a description of the received twist velocities. The twist velocities are split into linear (x, y, z) and angular (z) components. The linear components are mostly (0.00, 0.00, 0.00), while the angular component is 1.00. The messages are repeated several times, with some showing a change in the angular component to 0.00.

```
Received Twist Velocities -> Lin
[ INFO] [1720524521.903573884]: Arduino Log: ear (x,y,z): (0.00,1.00,0.00) Angul
ar (z): 1.00

[ INFO] [1720524522.003391300]: Arduino Log: Received Twist Velocities -> Linear
(x,y,z): (0.00,0.00,0.00) Angular (z): 1.00

[ INFO] [1720524522.803953835]: Arduino Log: Received Twist Velocities -> Linear
(x,y,z): (0.00,0.00,0.00) Angular (z): 1.00

[ INFO] [1720524522.903950531]: Arduino Log: Received Twist Velocities -> Linear
(x,y,z): (0.00,0.00,0.00) Angular (z): 1.00

[ INFO] [1720524523.003394139]: Arduino Log: Received Twist Velocities -> Linear
(x,y,z): (0.00,0.00,0.00) An
[ INFO] [1720524523.103418703]: Arduino Log: gular (z): 1.00
Received Twist Velocities -> Lin
[ INFO] [1720524523.203392745]: Arduino Log: ear (x,y,z): (0.00,1.00,0.00) Angul
ar (z): 0.00
Received Twist Velocities -> Lin
[ INFO] [1720524523.303393506]: Arduino Log: ear (x,y,z): (0.00,0.00,0.00) Angul
ar (z): 1.00
```

Fig 33.(B)

9.4 Interfacing Jetson Nano and Arduino Uno

The interface between the Jetson Nano and Arduino Uno is established using a serial connection as follows:

- **Serial Communication Setup:** The Jetson Nano uses the “pySerial” library to open a serial port connection to the Arduino Uno. The connection parameters, such as baud rate, data bits, stop bits, and parity, are configured to match the settings on the Arduino Uno. This ensures stable communication between the two devices.
- **Sending Commands:** The Jetson Nano publishes velocity commands to the `cmd_vel` topic. These commands are in the form of linear and angular velocities. The Arduino Uno, subscribed to this topic, reads these commands and converts them into PWM signals for the motor driver.
- **Arduino Code:** On the Arduino side, the `Serial.read()` function is used to receive commands. The code snippet below shows how the Arduino processes incoming velocity commands and sets the PWM signals accordingly.

9.5 Block Diagram.

Figure 34. the block diagram of our project's working, illustrating the key components and their interconnections for the PureZone: Autonomous UV and Fog Disinfection Rover.

1. **NVIDIA Jetson Nano:** Main processing unit, connected to sensors and modules.
2. **Intel T265 Tracking Camera:** Provides visual tracking.
3. **RPLidar A2m7:** Offers distance measurements and object detection.
4. **UV Module:** Controlled by the Arduino UNO for disinfection.
5. **Arduino UNO:** Manages communication and controls the motor drivers and disinfection modules.
6. **Motor Control Circuit:** Interfaces with IBT-2 motor drivers.
7. **IBT-2 Motor Drivers:** Control two sets of three DC motors.
8. **DC Motors:** Enable rover movement.
9. **Battery 12V 6A:** Powers the entire system.
10. **Fog Module:** Controlled by the Arduino UNO for fog-based disinfection.

Sensors send data to the Jetson Nano, which processes it and communicates with the Arduino UNO. The Arduino UNO controls the motors and disinfection modules, powered by the battery, enabling autonomous navigation and disinfection.

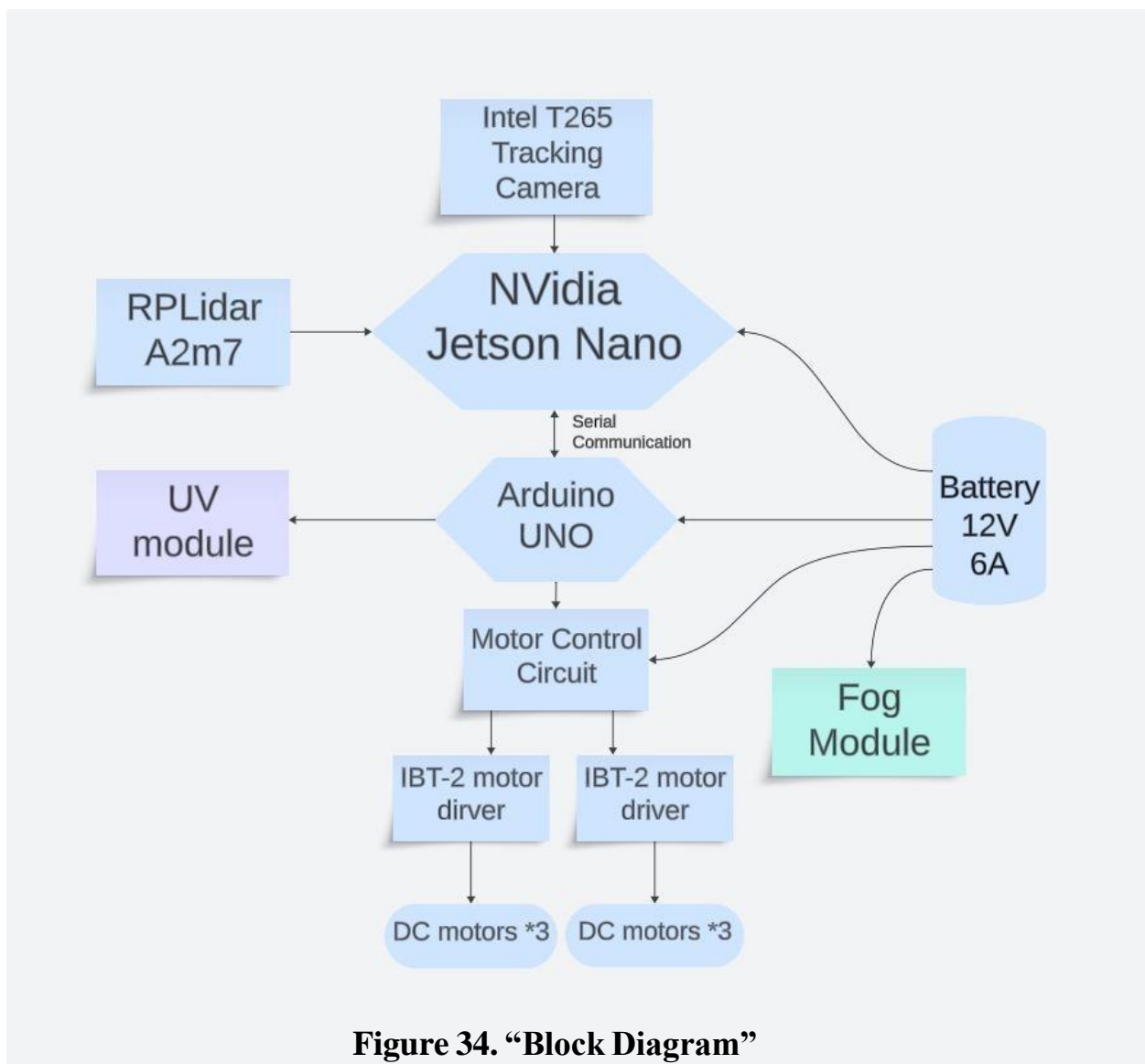


Figure 34. “Block Diagram”

9.6 UV Disinfection System

Our rover utilizes UV (Ultraviolet) light-emitting diodes (LEDs) for targeted disinfection. LEDs offer several advantages for our application compared to traditional UV tubes:

- **Compact Size:** LEDs are significantly smaller and lighter than UV tubes, making them ideal for our compact rover design.
- **Durability:** LEDs are more resistant to shock and vibration, reducing the risk of damage during operation.
- **Directional Control:** LEDs can be easily focused and directed, allowing for targeted disinfection of specific areas.

In our implementation, we've opted for a windmill-shaped arrangement of UV LEDs soldered on a breadboard and connected in series. This configuration maximizes the coverage area while maintaining a compact form factor. The entire UV LED assembly is mounted on a servo motor controlled by the Arduino Uno. By manipulating the servo motor position, we can control the direction of the UV light and ensure effective disinfection of targeted areas.

Effectiveness of UV Light Disinfection:

Scientific research has shown that UV light can be effective in inactivating a wide range of microorganisms, including bacteria, viruses, and fungi. Studies have demonstrated that UV light at specific wavelengths (primarily UVC radiation between 250-275 nm) can disrupt the DNA or RNA of these pathogens, rendering them unable to replicate and spread infection. Effectiveness depends on various factors, including:

- **Wavelength:** UVC light is generally considered the most germicidal wavelength range.
- **Exposure Time:** The longer the exposure to UV light, the greater the germicidal effect.
- **Dosage:** This is a combination of intensity (measured in microwatts per square centimeter, $\mu\text{W}/\text{cm}^2$) and exposure time.
- **Distance from Light Source:** The intensity of UV light decreases with distance from the source.
- **Surface Material:** Certain materials may reflect or absorb UV light, impacting its effectiveness.

9.7 Fog Disinfection System

Our rover utilizes an ultrasonic fogging system for targeted disinfection. Ultrasonic foggers work by creating a fine mist of disinfectant solution using high-frequency vibrations. This mist can effectively penetrate cracks and crevices, reaching areas that may be inaccessible to traditional cleaning methods. Here's how the system integrates:

- **Ultrasonic Fogger:** The chosen ultrasonic fogger will be connected to the Arduino Uno. The Arduino Uno can control the activation and duration of the fogger based on pre-programmed disinfection zones or manual commands.
- **Disinfectant Solution:** The choice of disinfectant solution will depend on the target pathogens and safety considerations. Common options include:
 - **Quaternary Ammonium Compounds (Quats):** These are broad-spectrum disinfectants effective against bacteria, viruses, and some fungi. However, they can be corrosive to certain materials and may not be suitable for all environments.
 - **Hydrogen Peroxide:** This is a versatile disinfectant that can be effective against a wide range of pathogens. However, it can have a bleaching effect on some fabrics and may irritate the respiratory system at high concentrations.
- **Safety Considerations:** When using fog disinfection systems, it's crucial to follow the manufacturer's instructions for safe operation and handling of the chosen disinfectant solution. The rover may need to be equipped with ventilation mechanisms to remove residual fog after disinfection cycles.

Effectiveness of Fog Disinfection:

Fog disinfection can be an effective tool for reducing the bioburden (microbial load) in an environment. However, its effectiveness depends on several factors, including:

- **Selection of Disinfectant:** The chosen disinfectant solution must be effective against the target pathogens.
- **Fogging Technique:** Proper coverage and penetration of the fog mist are essential for successful disinfection.
- **Environmental Conditions:** Factors like humidity and temperature can impact the efficacy of fog disinfection.
- **Organic Material:** Organic matter (e.g., blood, dirt) can deactivate some disinfectants, reducing their effectiveness.

9.8 Integration and Control System

The Jetson Nano acts as the central processing unit, coordinating all functionalities. It communicates with the Arduino Uno via serial communication to send high-level commands for:

- **Motor Control:** The Arduino Uno translates velocity commands from the navigation stack into motor control signals.
- **UV Light Activation:** The Arduino Uno controls the activation and direction of the UV LED assembly based on pre-programmed disinfection zones or manual commands.
- **Fog System Activation:** The Arduino Uno controls the activation and duration of the ultrasonic fogger based on the disinfection plan.

9.9 Exploration Server and Bash File Integration

The Exploration Server manages autonomous exploration tasks, defining exploration goals and planning routes based on the map generated by Hector SLAM. It ensures the rover navigates efficiently, avoids obstacles, and optimizes area coverage. By processing real-time sensor data, the Exploration Server dynamically adapts the exploration strategy, enhancing the rover's ability to autonomously cover designated areas and achieve targeted disinfection.

The bash file streamlines the setup and execution of system components by automating several tasks. It handles the initialization of ROS nodes, configuration of parameters, and launching of essential processes. This script reduces manual setup time and ensures consistency in the execution of software components, supporting a seamless and efficient operational workflow for the rover.

9.10 Conclusion

This chapter explored the practical implementation details of our autonomous UV and fog disinfection rover. We discussed how the Jetson Nano serves as the central processing unit, the RP Lidar and tracking device provide essential sensor data, and the Arduino Uno interfaces with the motor driver for movement control. We also explored the integration details of the UV LED system using a servo motor for targeted disinfection and the ultrasonic fogging system with considerations for disinfectant selection and safety. The addition of the Exploration Server and bash file ensures efficient autonomous operation and simplifies the setup process. The successful integration of these hardware and software components, along with the capability for manual control via teleop twist commands, paves the way for a robust and effective autonomous disinfection solution.

Chapter 10

Applications of the Autonomous UV Disinfection Robot

10.1 Introduction

The autonomous UV disinfection robot is designed to enhance cleanliness and hygiene in various environments by combining UV light disinfection, fog-based disinfection, autonomous mapping, and navigation technologies. This chapter explores its applications across different sectors and examines the technical aspects that enable its efficient operation.

10.2 Applications in Various Sectors

The autonomous UV disinfection robot is versatile and can be deployed in multiple settings to ensure high levels of hygiene. In healthcare facilities, it disinfects patient rooms, operating theatres, and isolation wards to prevent hospital-acquired infections. In commercial and public spaces, such as office buildings, shopping malls, and retail stores, it continuously disinfects high-touch surfaces. Educational institutions benefit from its use in classrooms, laboratories, libraries, and study areas to maintain hygiene. Transportation hubs, including terminals and waiting areas, and vehicles, can be disinfected efficiently. In the hospitality industry, it ensures guest rooms, lobbies, and dining areas are safe for guests. Additionally, manufacturing and industrial facilities, particularly in cleanrooms and laboratories, can maintain sterility with the robot's use.

10.3 Autonomous Mapping

10.3.1 RP Lidar A2M7 Model

The RP Lidar A2M7 model from SLAMTECH is integral for mapping. This advanced Lidar provides precise distance measurements, enabling the robot to create detailed maps of its environment.

10.3.2 Mapping Process

Initial Data Collection: The RP Lidar collects thousands of data points per second as the robot moves through its environment. This data represents the distances to various objects and surfaces around the robot.

Sequential Package Execution: The mapping process begins with running the RP Lidar package to gather raw data. Following this, the tracking device package processes the data to provide positional information. Finally, Hector SLAM (Simultaneous Localization and Mapping) combines these inputs to create a cohesive map.

Real-Time Mapping: During operation, the robot generates maps in real-time, identifying obstacles (represented in black in the map) and free spaces. The RP Lidar scans continuously, updating the map dynamically as the robot moves. Only one instance of RVIZ (ROS visualization tool) runs at a time to display the generated map. If navigation RVIZ runs, it closes the Hector SLAM RVIZ automatically to avoid conflicts.

10.3.3 Real-Time Updates

Simultaneous Localization and Mapping (SLAM): Hector SLAM is utilized for real-time map generation and localization. It updates the robot's position on the map concurrently, ensuring precise and up-to-date navigation paths.

10.4 Autonomous Navigation

10.4.1 Navigation Algorithms

Path Planning: The robot uses sophisticated path-planning algorithms to determine the most efficient and safe routes to its destinations. These algorithms take into account the current map, obstacle locations, and the desired end point.

Obstacle Avoidance: The robot dynamically adjusts its path to avoid newly detected obstacles. It uses sensor data to detect objects and recalculates its route to bypass them.

10.4.2 Camera-Based Odometry

Odometry Data: The camera provides odometry data, which is essential for tracking the robot's movement and position changes over time. This data includes the distance travelled and changes in orientation.

Integration with Lidar: The camera and Lidar work together to offer precise localization. While the Lidar maps the environment, the camera ensures accurate tracking of the robot's movement within this map.

10.4.3 Navigation Process

Sequential Package Execution: After the mapping phase, the navigation package runs, utilizing data from both the camera and Lidar. This involves running the camera package to gather odometry data, followed by the navigation algorithm to calculate the robot's movement.

Command Velocities: The navigation system generates command velocities, which include linear velocity (x) for forward movement and angular velocity (z) for turning. These commands are sent to an Arduino, which converts them into control signals for the motor driver.

10.4.4 Goal Setting and Achievement

Manual Goal Setting: Operators can manually set waypoints using a graphical interface. The robot navigates to these points, updating the map simultaneously. As the robot moves towards its goal, it continues to map the environment, ensuring it adapts to any changes.

Autonomous Goal Achievement: The robot autonomously reaches set goals, adjusting for any changes in the environment. When a goal is achieved, new waypoints can be set for continuous operation. The TF (Transform) tree structure shows data flow and support relationships between different components, ensuring coordinated operation.

10.5 Technical Aspects

10.5.1 Sensor Suite

Lidar: The RP Lidar A2M7 provides accurate distance measurements for mapping and navigation.

Camera: Used for odometry and obstacle detection, ensuring precise localization.

10.5.2 Software and Hardware Integration

Jetson Nano: This powerful computing module from NVIDIA runs a custom version of Ubuntu 18.04 with ROS Melodic. It processes sensor data and executes the necessary algorithms for mapping and navigation.

Arduino Integration: The Arduino microcontroller receives command velocities from the Jetson Nano and controls the motor driver accordingly.

10.5.3 Power Management System

Battery Monitoring: Continuously tracks battery status to ensure efficient operation and longevity.

Power Distribution: Manages power supply to all robot components, ensuring stable operation.

10.5.4 Battery Management System

Safety Features: Protects against overcharge, over-discharge, and overheating, ensuring safe operation.

Cell Balancing: Ensures even charging and discharging of battery cells to maximize lifespan.

State Estimation: Provides accurate estimates of remaining charge, state of health, and state of power.

10.6 Wireless Communication: Wi-Fi and Bluetooth

Wi-Fi: Used for remote monitoring, control, data transfer, and software updates. It enables the robot to communicate with cloud-based platforms and other systems in smart environments.

Bluetooth: Facilitates local control, data exchange, and initial setup. It provides a secondary communication channel for redundancy and fallback.

10.6.1 Automatic Report Generation

Data Collection: Logs sensor data, odometry, and disinfection activities throughout its operation.

Report Compilation: Aggregates collected data into comprehensive reports, including performance metrics and visual representations of disinfection coverage and effectiveness.

Delivery and Storage: Reports are uploaded to the cloud and accessible via a user-friendly dashboard, allowing operators to review the robot's performance and compliance with hygiene standards.

10.7 Conclusion

The autonomous UV disinfection robot's versatile applications highlight its potential to revolutionize hygiene standards across various industries. By integrating advanced technologies such as autonomous mapping, path exploration, and navigation, this robot not only enhances disinfection efficacy but also improves operational efficiency and safety. As technology continues to evolve, the role of autonomous disinfection robots will become increasingly vital in ensuring clean and safe environments for all.

Appendix 1

Arduino Code for Motor Control:

```
#include <Servo.h>

// Initialize variables for linear and angular velocities
float linear_x = 0.0;
float linear_y = 0.0;
float linear_z = 0.0;
float angular_z = 0.0;

// Define pin numbers for Motor 1
const int RPWM1 = 6; // RPWM pin for Motor 1
const int LPWM1 = 5; // LPWM pin for Motor 1

// Define pin numbers for Motor 2
const int RPWM2 = 10; // RPWM pin for Motor 2
const int LPWM2 = 9; // LPWM pin for Motor 2

Servo myservo; // create servo object to control a servo
int pos = 0; // variable to store the servo position

void setup() {
    // Initialize serial communication
    Serial.begin(9600);

    // Initialize motor control pins as outputs
    pinMode(RPWM1, OUTPUT);
    pinMode(LPWM1, OUTPUT);
    pinMode(RPWM2, OUTPUT);
```

```

pinMode(LPWM2, OUTPUT);

// Start with motors stopped
stopMotor1();
stopMotor2();

// Attach the servo to pin 8
myservo.attach(8);
}

void loop() {
    // Check if there's data available to read from the serial port
    if (Serial.available() > 0) {
        // Read the incoming byte
        char incomingByte = Serial.read();

        // Check if the incoming byte is the start of the message
        if (incomingByte == '<') {
            // Read the values for linear and angular velocities
            while (Serial.available() < 14); // Wait for complete message
            linear_x = Serial.parseFloat();
            angular_z = Serial.parseFloat();
            linear_y = Serial.parseFloat();
            linear_z = Serial.parseFloat();

            // Read the end of message character
            while (Serial.available() > 0 && Serial.read() != '>');

            // Log the received values
            logReceivedValues();
        }
    }
}

```

```

    // Control the motors based on the received values
    controlMotors(linear_x, angular_z);
}
}

// Servo control loop
for (pos = 0; pos <= 50; pos += 1) { // goes from 0 degrees to 45 degrees
    myservo.write(pos);           // tell servo to go to position in variable 'pos'
    delay(15);                     // waits 15 ms for the servo to reach the position
}
for (pos = 50; pos >= 0; pos -= 1) { // goes from 45 degrees to 0 degrees
    myservo.write(pos);           // tell servo to go to position in variable 'pos'
    delay(15);                     // waits 15 ms for the servo to reach the position
}

// Add delay if needed to prevent flooding the serial port
delay(100);
}

// Function to control the motors based on linear and angular velocities
void controlMotors(float linearX, float angularZ) {
    int motor1Speed = 0;
    int motor2Speed = 0;
    int direction1 = 1;
    int direction2 = 1;

    // Determine which value is larger (in absolute terms) and set motor speeds accordingly
    if (abs(linearX) > abs(angularZ)) {
        // Use linearX to set the motor speeds
        if (linearX >= 0) {
            if (linearX <= 0.11 && linearX >= 0.03) {

```

```

    motor1Speed = int(255 * 0.65); // 75% of max speed
    motor2Speed = int(255 * 0.65); // 75% of max speed
} else if (linearX >= 0.3 && linearX <= 0.35) {
    motor1Speed = int(255 * 0.65); // 85% of max speed
    motor2Speed = int(255 * 0.65); // 85% of max speed
} else if (linearX >= 0.2 && linearX < 0.3) {
    motor1Speed = int(255 * 0.60); // 80% of max speed
    motor2Speed = int(255 * 0.60); // 80% of max speed
} else {
    motor1Speed = int(linearX * 350); // Normal scaling factor
    motor2Speed = int(linearX * 350); // Normal scaling factor
}

direction1 = 1; // Forward
direction2 = 1; // Forward
} else {
    if (linearX >= -0.11 && linearX <= -0.03) {
        motor1Speed = int(255 * 0.65); // 75% of max speed
        motor2Speed = int(255 * 0.65); // 75% of max speed
    } else if (linearX >= -0.35 && linearX <= -0.3) {
        motor1Speed = int(255 * 0.65); // 85% of max speed
        motor2Speed = int(255 * 0.65); // 85% of max speed
    } else if (linearX >= -0.3 && linearX < -0.2) {
        motor1Speed = int(255 * 0.60); // 80% of max speed
        motor2Speed = int(255 * 0.60); // 80% of max speed
    } else {
        motor1Speed = int(-linearX * 350); // Normal scaling factor
        motor2Speed = int(-linearX * 350); // Normal scaling factor
    }

    direction1 = -1; // Backward
    direction2 = -1; // Backward
}

```

```

} else {

    // Use angularZ to set the motor speeds

    if (angularZ >= 0) {

        if (angularZ <= 0.11 && angularZ >= 0.03) {

            motor1Speed = int(255 * 0.65); // 75% of max speed

            motor2Speed = int(255 * 0.65); // 75% of max speed

        } else if (angularZ >= 0.3 && angularZ <= 0.35) {

            motor1Speed = int(255 * 0.65); // 85% of max speed

            motor2Speed = int(255 * 0.65); // 85% of max speed

        } else if (angularZ >= 0.2 && angularZ < 0.3) {

            motor1Speed = int(255 * 0.60); // 80% of max speed

            motor2Speed = int(255 * 0.60); // 80% of max speed

        } else if (angularZ > 0.9) {

            motor1Speed = 0;

            motor2Speed = 0;

        } else {

            motor1Speed = int(angularZ * 255); // Normal scaling factor

            motor2Speed = int(angularZ * 255); // Normal scaling factor

        }

        direction1 = 1; // Forward for Motor 1

        direction2 = -1; // Backward for Motor 2

    } else {

        if (angularZ >= -0.11 && angularZ <= -0.03) {

            motor1Speed = int(255 * 0.65); // 75% of max speed

            motor2Speed = int(255 * 0.65); // 75% of max speed

        } else if (angularZ >= -0.35 && angularZ <= -0.3) {

            motor1Speed = int(255 * 0.65); // 85% of max speed

            motor2Speed = int(255 * 0.65); // 85% of max speed

        } else if (angularZ >= -0.3 && angularZ < -0.2) {

            motor1Speed = int(255 * 0.60); // 80% of max speed

            motor2Speed = int(255 * 0.60); // 80% of max speed

        }

    }

}

```

```

    } else if (angularZ < -0.9) {
        motor1Speed = 0;
        motor2Speed = 0;
    } else {
        motor1Speed = int(-angularZ * 255); // Normal scaling factor
        motor2Speed = int(-angularZ * 255); // Normal scaling factor
    }

    direction1 = -1; // Backward for Motor 1
    direction2 = 1; // Forward for Motor 2
}

setMotor1(motor1Speed, direction1);
setMotor2(motor2Speed, direction2);
}

void setMotor1(int speed, int direction) {
    // Ensure speed is within 0-255 range
    speed = constrain(speed, 0, 255);

    if (direction == 1) {
        // Forward
        analogWrite(RPWM1, speed);
        analogWrite(LPWM1, 0);
    } else if (direction == -1) {
        // Reverse
        analogWrite(RPWM1, 0);
        analogWrite(LPWM1, speed);
    }
}
}

```



```

void setMotor2(int speed, int direction) {
    // Ensure speed is within 0-255 range
    speed = constrain(speed, 0, 255);

    if (direction == 1) {
        // Forward
        analogWrite(RPWM2, speed);
        analogWrite(LPWM2, 0);
    } else if (direction == -1) {
        // Reverse
        analogWrite(RPWM2, 0);
        analogWrite(LPWM2, speed);
    }
}

void stopMotor1() {
    analogWrite(RPWM1, 0);
    analogWrite(LPWM1, 0);
}

void stopMotor2() {
    analogWrite(RPWM2, 0);
    analogWrite(LPWM2, 0);
}

// Function to log the received values
void logReceivedValues() {
    String logMessage = "Received Twist Velocities -> Linear (x,y,z): (" + String(linear_x)
+ "," + String(linear_y) + "," + String(linear_z) + ") Angular (z): " + String(angular_z);
    Serial.println(logMessage);
}

```

References

- [1] R. Zhang, Y. Li, A. L. Zhang, Y. Wang, and M. J. Molina, "Identifying airborne transmission as the dominant route for the spread of covid19," *Proceedings of the National Academy of Sciences*, vol. 117, no. 26, pp. 14 857–14 863, 2020.
- [2] S. W. X. Ong, Y. K. Tan, P. Y. Chia, T. H. Lee, O. T. Ng, M. S. Y. Wong, and K. Marimuthu, "Air, surface environmental, and personal protective equipment contamination by severe acute respiratory syndrome coronavirus 2 (sars-cov-2) from a symptomatic patient," *Jama*, vol. 323, no. 16, pp. 1610–1612, 2020.
- [3] I. Kalinov, E. Safronov, R. Agishev, M. Kurenkov, and D. Tsetserukou, "High-precision uav localization system for landing on a mobile collaborative robot based on an ir marker pattern recognition," in *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*. IEEE, 2019, pp. 1–6.
- [4] I. Kalinov, A. Petrovsky, V. Ilin, E. Pristanskiy, M. Kurenkov, V. Ramzhaev, I. Idrisov, and D. Tsetserukou, "Warevision: Cnn barcode detection-based uav trajectory optimization for autonomous warehouse stocktaking," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6647–6653, 2020.
- [5] P. Karpyshev, V. Ilin, I. Kalinov, A. Petrovsky, and D. Tsetserukou, "Autonomous mobile robot for apple plant disease detection based on cnn and multi-spectral vision system," in *2021 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 2021, pp. 157–162.
- [6] R. Bormann, J. Hampp, and M. Hagele, "New brooms sweep clean- "an autonomous robotic cleaning assistant for professional office cleaning," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 4470–4477.
- [7] Aladin Begic, "Application Of Service Robots For Disinfection In Medical Institutions", *Advanced Technologies, Systems, and Applications II*. 2018; 28: 1056–1065
- [8] G Katara, N Hemvani, S Chitnis, V Chitnis, "Surface Disinfection By Exposure To Germicidal Uv Light", *Indian Journal of Medical Microbiology* (2008)26(3):241-42.
- [9] DISINFECTION OF AIRBORNE VIRUSES USING UV
<https://spectrum.ieee.org/techtalk/biomedical/devices/ultraviolet-revolution-could-faruv-lightprovide-widespread-safe-disinfection-of-airborne-viruses>
- [10] E.C. Friedberg, G.C. Walker, W. Siede, R.D. Wood, R.A. Schultz, T. Ellenberger, "DNA REPAIR AND MUTAGENESIS," *ASN Press*, Washington, 2006.
- [11] Anshika Chaturvedi, Praveen Kumar, Seema Rawat, "Proposed Noval Security System Based On Passive Infrared Sensor", 2016 International Conference on Information Technology (InCITE).
- [12] <https://spectrum.ieee.org/techtalk/semiconductors/optoelectronics/ultraviolet-led-maker-demonstrates-30-second-coronavirus-kill>

- [13] O. Urfaliglu, Emin B. Soyer, B. Ugur Toreyin, A. Enis Cetin, "Pir-Sensor Based Human Motion Event Classification", 2008 IEEE 16th Signal Processing, Communication and Applications Conference.
- [14] Jaeseok Yun and Sang-Shin Lee, "Human Movement Detection And Identification Using Pyroelectric Infrared Sensors", Embedded Software Convergence Research Center, Korea Electronics Technology Institute, 25 Saenari-ro, Bundang-gu, Seongnam 463070, Korea, 5 May 2014.
- [15] https://www.researchgate.net/publication/327965269_Line_Follower_Robot_For_Industrial_Manufacturing_Process.
- [16] Kazi Mahmud Hasan, Abdullah-Al-Nahid, Abdullah Al Mamun, "Implementation of Autonomous Line Follower Robot for Various Purposes ", 2013 IEEE.
- [17] <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC406306>
- [18] Stepan Perminov, Nikita Mikhailovskiy, Alexander Sedunin, Iaroslav Okunevich, Ivan Kalinov, Mikhail Kurenkov, and Dzmitry Tsetserukou, "UltraBot: Autonomous Mobile Robot for Indoor UV-C Disinfection" ,2021 IEEE.
- [19] UV Disinfection Robots: A Review Ishaan Mehta,a,1 Hao-Ya Hsueh,a,*,1 Sharareh Taghipour,a Wenbin Li,b and Sajad Saeedia
- [20] <https://www.ebay.com.au/itm/295686685721>
- [21] <https://www.amazon.sa/-/en/youyeetoo-Scanning-brushless-Avoidance-Navigation/dp/B082FD34R5>
- [22] <https://www.socketxp.com/iot/nvidia-jetson-nano-vs-raspberry-pi-which-one-is-better-for-your-project/>
- [23] <http://dienmayviet.com.vn/intel-realsense-tracking-camera-t265-1270082.html>

