



**Politecnico
di Torino**

Politecnico di Torino

Master of Science in Mechatronics Engineering

A.A. 2025/2026

Graduation Session March 2026

CO₂-Guided Navigation for Autonomous Sanitization Robotic Platforms

Supervisor:

Marcello Chiaberge
Mauro Martini

Candidate:

Davide Cappelletto

Abstract

Service robotics is a rapidly growing field within robotics that focuses on the development of systems designed to assist humans in non-industrial settings. An important trend within this field is the deployment of such systems in health-related applications. After the SARS-CoV-2 pandemic, awareness of the need for effective environmental monitoring and sanitization of indoor environments resurfaced, to minimize the risks of pathogens and other dangerous particles spread, while ensuring the well-being of the inhabitants of public spaces and workplaces. Given the fact that carbon dioxide (CO₂) is largely considered a suitable biomarker for the evaluation of Indoor Air Quality (IAQ) and human presence, this thesis presents the development of a CO₂-informed, waypoint-based, navigation algorithm exploiting a behavior-tree-based approach integrated in a heavy-load, multi-wheeled platform able to carry a mobile sanitizer. The navigation system is an implementation of Nav2's BT framework, with a server managing a set of client calls, each related to a BT node. The platform is completely custom-made and employs an Intel NUC to manage a set of peripherals and a micro-controller for wheel actuation. The system performs an autonomous patrolling activity through the given waypoints that prioritizes nodes with higher CO₂ concentration to sanitize first, limiting the risk of infections and improving IAQ, while constantly delivering a heatmap of the CO₂ distribution inside the surveyed environment. On field tests, the platform reported higher CO₂ concentration in populated areas, resulting in frequent monitoring of related waypoints. Limitations and future implementations are discussed, including the implementation of multiple, inherent sensors.

Acknowledgements

I would like to thank my family, who never made me feel lost or alone, for the immense support and love with which they filled me during this exciting yet perilous journey.

A special thanks goes to Elisa, my guiding light, who kept me on the right path during these years, even when many kilometers of rails kept us apart. Hopefully, that won't be the case anymore.

I would also like to thank my friends and colleagues, with whom I shared joys and fears, and who made me realize that each of us has our own journey in life to pursue.

Last but not least, I would like to thank the staff of PIC4SeR, who gave me the opportunity to work on this project and with whom I enjoyed collaborating and learning throughout this entire experience. Thank you, Andrea, Gianluca, and Mauro.

Table of Contents

List of Tables	VI
List of Figures	VII
1 Introduction	1
1.1 Problem Statement and Contributions	3
1.2 Structure	4
2 Background & Related Work	5
2.1 Sanitization Techniques in Service Robotics	5
2.2 Environmental Sensing	10
2.3 Map-based navigation	16
2.4 Behavior-based navigation	18
2.5 Robot Operating System 2	21
3 Methodology	22
3.1 System Overview and Requirements	23
3.2 Robotic Platform	24
3.2.1 Sensors	24
3.2.2 Actuators	28
3.2.3 Interfaces	30
3.2.4 Sanitization Unit	31

3.2.5	Operative Unit	31
3.3	Software Architecture	33
3.3.1	Sensor integration and CO ₂ measurement	33
3.3.2	CO ₂ -Based BT Navigation Package	34
3.3.3	Runtime Integration: Execution, Supervision, and Backend .	39
3.3.4	Interface (<code>button_publisher</code>)	39
3.3.5	Composable strategy	44
4	Experimental Evaluation	46
4.1	Simulated Experiments - first approach	46
4.2	Simulated Experiments - second approach	52
4.3	Real Deployment	54
5	Conclusion and Future Work	57
A	Probabilistic Localization Background	59
A.1	Particle Filters: A Nonparametric Approximation	61
A.2	Monte Carlo Localization	63
A.3	Adaptive Monte Carlo Localization	64
B	<i>WaypointsBattery</i> - Complete Tree	66
	Bibliography	68

List of Tables

2.1	Summary of both internal and leaf nodes.	19
2.2	ROS 2 Communication Primitives	21
3.1	Functional System Requirements	23
3.2	RPLIDAR A1 Sample Point Data Information	26
3.3	RPLIDAR A1M8 Technical Specifications	26
3.4	Operating range	27
3.5	Characteristic parameters of the RoboMaster M3508 P19 motor . .	29
3.6	M3508 P19 performance parameters using the C620 speed controller at rated voltage	29
4.1	Waypoints coordinates	48
4.2	Simulated CO ₂ -informed navigation, first cycle.	50
4.3	Simulated CO ₂ -informed navigation, second cycle.	51
4.4	Waypoints coordinates in ISPIC's offices.	55

List of Figures

2.1	UV-C disinfection robots.	6
2.2	Autonomous Indoor Cleaning Robot (AICR).	8
2.3	Example of monitoring robotic platforms.	12
2.4	Mobile Measuring Robot detailed in [29].	14
2.5	An example scheme of ports/blackboard interaction [42].	20
3.1	Robotic platform’s diagram.	24
3.2	Robotic platform designed for the project.	25
3.3	RPLiDAR A1 mounted on the platform.	25
3.4	CO ₂ sensor (Xensiv PAS CO2 1.5) and its protective case.	27
3.5	Representation of <i>Robomaster</i> ’s brushless motor.	28
3.6	Example of button panel peripheral used for the platform.	30
3.7	ASPRA portable sanitizer mounted on the robot [49].	31
3.8	<i>WaypointsBattery</i> subtree.	37
3.9	<i>GoToDock</i> subtree.	39
3.10	Overall system’s structure and main interactions.	44
3.11	Compose structure.	45
4.1	World created in <i>Gazebo</i> for the simulation.	47
4.2	URDF robot model.	47
4.3	2D map of the simulated office environment.	48

4.4	Visual representation of waypoints on the 2D map. "0" indicates the origin [0,0].	49
4.5	chopper platform navigating through the environment, approaching waypoint 6. Heatmap of the first 4 waypoints already created. . . .	50
4.6	<code>bt_server_node</code> sorting waypoint list and evaluating <i>Waypoint 10</i> as first waypoint to explore in second cycle.	51
4.7	<code>bt_server_node</code> sorting waypoint list and evaluating <i>Waypoint 2</i> as first waypoint to explore in third cycle.	51
4.8	Simulation map divided in 4 quadrants. Every quadrant is associated with a CO ₂ value (in ppm).	52
4.9	Overall caption for the 4 stacked subfigures	53
4.10	Heatmap processed by the server node after the exploration of the 12 th waypoint.	53
4.11	Map scan of ISPIC's offices.	54
4.12	Visual representation of waypoints in the office.	55
4.13	Heatmap of ISPIC's working space.	56

Chapter 1

Introduction

As technology infiltrates evermore deeply into our daily lives, the opportunity to make people’s lives easier, or even safer, has become a reality, by delivering systems that are designed to be intuitive and efficient for the purpose of enhancing productivity and safety, lightening workloads and improving overall well-being.

Service robotics is a rapidly growing field within robotics that focuses on the development of robots designed to assist humans in non-industrial settings. The International Organization for Standardization (ISO) defines a service robot as one “in personal use or professional use that performs useful tasks for humans or equipment” [1]. In contrast to industrial robots, which are typically confined to structured factory environments, service robots operate in homes, offices, hospitals, warehouses, and other human-centered spaces.

Recent systematic reviews argue that service robotics has transitioned from a niche research area into a major branch of robotics, driven by advances in perception, AI, and networked systems, and by demographic and economic pressures such as labor shortages and aging populations [2, 3]. Within this broad category, service robots are commonly classified along two main axes [2]:

- **Professional vs. Domestic**, depending on whether they are used in commercial or home settings;
- **Social vs. Non-social**, depending on whether human–robot interaction is central to their function.

This yields four high-level types: professional non-social (e.g., cleaning, logistics, inspection), professional social (e.g., hotel guides, reception robots), domestic non-social (e.g., vacuum cleaners, lawn mowers), and domestic social robots (e.g., companions, elder-care assistants). Furthermore, market analyses confirm that

this field represents a significant and fast-growing segment of the global robotics industry: a recent report estimates that the global service robotics market was around USD 36.1 billion in 2024 and is projected to reach roughly USD 166.6 billion by 2034, corresponding to a compound annual growth rate of about 17%. Growth is particularly strong in healthcare, logistics/warehousing, and domestic cleaning, where robots address labor shortages, rising demand, and expectations for higher service quality. In the professional segment, sales of medical service robots have increased sharply, including surgical systems, rehabilitation and non-invasive therapy robots, and disinfection platforms. In domestic contexts, cleaning robots remain the dominant consumer application, and their integration with smart-home technologies is seen as a key direction for future products [3, 4].

From a technological perspective, recent surveys converge on three cross-cutting “foundations” that underlie most service robot applications [2, 3]:

1. **Perception, localization, and navigation** in unstructured environments;
2. **Human–robot interaction and interfaces**;
3. **Software architecture and platforms** for integrating heterogeneous components.

Navigation and mapping capabilities have been heavily influenced by the evolution of the Robot Operating System (ROS) ecosystem, in particular ROS 2 and the Nav2 stack, which now provide production-grade support for global planning, local trajectory control, layered costmaps, and behavior-tree-based task orchestration for a wide variety of service robots. On top of these foundations, modern service robots increasingly incorporate AI methods — machine learning for perception and decision-making, computer vision, and higher-level reasoning — to adapt their behavior to evolving tasks and environments [3, 5].

An important recent trend is the deployment of service robots in health-related and sanitization tasks in real environments. When in 2020 the world was struck by the SARS-CoV-2 (COVID-19) pandemic, a great awareness of the need for effective environmental monitoring and sanitization resurfaced: proper ventilation, surface disinfection, and air filtration were recognized as crucial measures to reduce the risk of infection in both public and private indoor spaces, where the vast majority of human activities occur (90%) [6], statistics increased during the pandemic due to lockdowns and subsequent remote work.

An essential fragment when evaluating and developing technologies and new solutions to counter or prevent the spread of pathogens resides in monitoring Indoor Air Quality (IAQ) which has then a direct impact on human health and can be affected by various chemicals, including gases like carbon dioxide (CO₂), Volatile Organic

Compounds (VOCs), particulate matter (PM), organic and inorganic contaminants, and biological particles such as bacteria and viruses [7]. In this context, particular relevance can be given to CO₂ which, although not being classified as a pollutant, it can be treated as such when its concentration compromises oxygen levels, to the point that it can also impair cognition [8]. Furthermore, this compound can be employed as a biomarker for ventilation adequacy, which is a critical factor in preventing the buildup of undesired gases [9], and in several papers it is also considered as a biomarker of airborne pathogen transmission [10, 11, 12, 13, 14]. This leads to the conclusion that the employment of CO₂ sensors is essential for monitoring IAQ. Implementing such monitoring systems in the design of a building is a complex and expensive task, particularly if implementing them in existing buildings.

For this reason, the use of mobile robotic platforms equipped with environmental sensors is emerging as a promising solution to provide flexible, scalable, and cost-effective monitoring and sanitization tasks. In literature, service robots implementing both monitoring and sanitization tasks, while present, are still relatively scarce, often proposing small platforms with limited autonomy that are mostly focused on proof-of-concept deployment than on real-world applications.

1.1 Problem Statement and Contributions

The overarching problem addressed in this thesis is how to deploy a heavy-duty, ROS 2-based mobile platform capable of both environmental monitoring and air sanitization in real indoor workplaces, using CO₂ as a proxy for ventilation adequacy and human presence. The goal is to design a navigation and control architecture that can prioritize poorly ventilated or highly occupied areas while remaining safe, scalable, and compatible with typical office constraints. In this context, the thesis contributes a heavy-duty, cost-effective, and flexible robotic platform that navigates in structured indoor environments following a behavior-tree-based control architecture, implementing CO₂-based environmental monitoring while performing air-filtration tasks to improve indoor air quality. The development of such system was requested by *Intesa San Paolo Innovation Center* (ISPIC) and developed at *PIC4SeR*.

The main contributions of this thesis are:

- A CO₂-informed waypoint navigation strategy that (i) acquires CO₂ measurements at each waypoint, (ii) reorders waypoint lists to start subsequent patrol cycles from the most CO₂-dense, and (iii) builds a real-time CO₂ heatmap using an exponential distance-weighted interpolation algorithm.

- Development of a ROS 2 and Nav2-based software architecture that combines Nav2's BT framework with a custom package including mission Behavior Trees, custom BT nodes, and a backend node for waypoint management, CO₂ sampling, and heatmap generation.
- System integration onto a heavy-duty, multi-wheeled robotic platform capable of operating a portable air-filtering unit, dimensioned for operation in office-like environments.
- Experimental evaluation in both Gazebo simulations and a real deployment in *Intesa San Paolo Innovation Center's* (ISPIC) offices, demonstrating autonomous patrolling, dynamic waypoint reordering based on CO₂ and continuous generation of CO₂ heatmaps during multi-cycle missions.

1.2 Structure

The thesis is structured as follows:

- **Chapter 2** covers fundamental concepts: navigation and mapping techniques for service robots, behavior trees for control architecture. The state of the art of service robotics is reviewed as well, with a particular focus on the application of robotic platforms for environmental monitoring and sanitization techniques;
- **Chapter 3** covers the CO₂-informed navigation system overview and its requirements, including the design of the robotic platform, the selection of sensors and actuators, and the idea behind the CO₂-based waypoint navigation strategy, and the developed software architecture to reach such requirements;
- **Chapter 4** presents the experimental evaluation of the system in a real-world indoor environment, assessing its navigation performance, environmental monitoring capabilities, and air filtration effectiveness;
- **Chapter 5** concludes the thesis, summarizing the main findings and outlining future research directions.

Chapter 2

Background & Related Work

2.1 Sanitization Techniques in Service Robotics

A variety of sanitization techniques have been explored in service robotics to clean surfaces and purify indoor air. These techniques can be broadly grouped into four categories: (i) ultraviolet-C (UV-C) germicidal irradiation, (ii) chemical or hybrid disinfection, (iii) mechanical air filtration (typically with HEPA or equivalent filters), and (iv) intelligent or sensor-driven purifiers that combine filtration with sensing and control. Each approach offers distinct advantages and limitations in terms of efficacy, safety, and suitability for operation in occupied indoor environments.

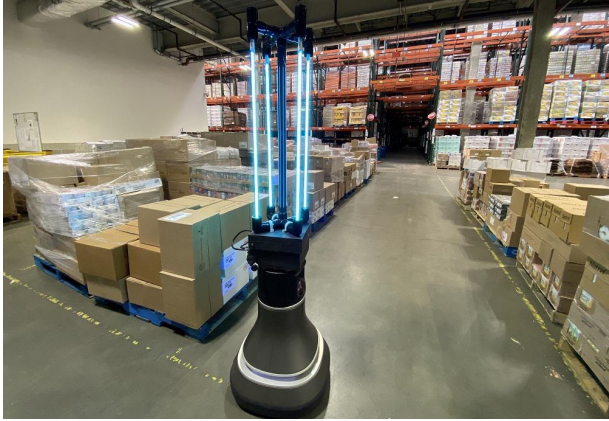
UV-C Germicidal Irradiation

UV-C disinfection robots use lamps emitting in the 200–280 nm range to inactivate microorganisms by damaging their DNA or RNA. A recent survey of UV-C disinfection robots for healthcare and public spaces shows a wide spectrum of platforms, from teleoperated carts to fully autonomous mobile robots, differing in lamp configurations, safety mechanisms, and navigation strategies [15]. These systems primarily target surface and near-field airborne disinfection in environments such as hospitals, offices, and warehouses.

A representative example is the mobile UVC disinfection robot deployed at the *Greater Boston Food Bank*, which combines a mobile base with a radial UV-C lamp assembly and coverage-based path planning [16]. The system models UV dose as a function of distance and exposure time, accounts for occlusions using LiDAR-derived visibility footprints, and plans nightly patrols that deliver at least 10 mJ/cm² to a 4000 ft² staging area in under 30 minutes. Experimental results suggest that such

doses are sufficient to neutralize on the order of 99% of coronaviruses on surfaces and in aerosols [16].

Similar principles are used in other indoor autonomous UV-C robots, which employ SLAM-based navigation and safety interlocks (e.g., human presence detection, door sensors) to ensure operation only in unoccupied rooms [17]. The key advantages of UV-C systems are their chemical-free operation, broad-spectrum antimicrobial efficacy, and the ability to disinfect complex surfaces and equipment that are unsuitable for liquid disinfectants. However, their main disadvantages are safety constraints—direct UV-C exposure is harmful to skin and eyes—and strong line-of-sight dependence. As a result, these robots typically operate in strictly unoccupied environments and must carefully account for occlusions and shadowed regions [15, 16, 17].



(a) UV-C Mobile Robot deployed at the *Greater Boston Food Bank*.



(b) RoboCoV Cleaner robot.

Figure 2.1: UV-C disinfection robots.

Hybrid UV-C and Chemical Disinfection

To mitigate line-of-sight limitations, some service robots combine UV-C irradiation with chemical aerosols. An example is the autonomous mobile combination disinfection system that integrates low-concentration hydrogen peroxide (H_2O_2) aerosol with UV-C lamps on a mobile robot platform [18]. In this system, UV-C radiation provides rapid disinfection of directly exposed surfaces, while the H_2O_2 aerosol penetrates into shadowed or occluded regions, leading to improved overall disinfection coverage. The authors report that the hybrid approach can achieve disinfection rates higher than either modality alone, at the cost of longer operation

times to allow for aerosol distribution and decay, and the need to control chemical residues and ventilation after treatment [18].

Hybrid systems thus trade increased complexity—chemical handling, ventilation requirements, and potentially longer room downtime—for improved coverage and reduced sensitivity to geometric occlusions. Like pure UV-C robots, they are generally constrained to operate without human occupants due to both UV and aerosol safety considerations.

Mechanical Air Filtration and HEPA-Based Systems

Mechanical air filtration devices, especially those employing HEPA or HEPA-equivalent filters, offer an alternative approach focused on continuous removal of airborne particles rather than surface disinfection. In educational settings, portable and mobile air purifiers have been evaluated as a means to reduce aerosol concentrations and the modeled risk of airborne disease transmission. For example, Curtius *et al.* tested four mobile HEPA purifiers in a school classroom and observed more than 90% reduction in aerosol particle number concentration within approximately 30 minutes when operated at suitable flow rates [19]. Their study also emphasizes that filtration does not reduce CO₂ levels, highlighting the need for separate ventilation or CO₂-based control [19].

Subsequent work on air filtration for schools has analyzed different deployment strategies (e.g., portable units in each classroom vs. central systems) and operational parameters (airflow rates, noise constraints), concluding that appropriately sized HEPA devices can significantly mitigate aerosol loads in classrooms and similar spaces [20]. Comparative studies between portable and large mobile air cleaners show trade-offs between flexibility (small, easily repositioned units), overall clean air delivery rate (CADR), and noise or energy consumption [21]. Large mobile units can provide higher CADR for big spaces but are bulkier and may be more intrusive, while multiple smaller units allow finer-grained placement but require more devices and coordination [21].

Although most HEPA-based systems in these studies are not full-fledged robots, they illustrate the core mechanism and its suitability for operation in occupied environments: unlike UV-C or chemical disinfection, HEPA filtration can be used continuously while people are present, as it does not introduce harmful radiation or aerosols. The main disadvantages are that filtration does not directly sanitize surfaces, requires periodic filter replacement and maintenance, and its effectiveness depends on room airflow patterns and correct sizing relative to room volume and occupancy [19, 20, 21].

Integrated HEPA and UV-C Purifiers on Mobile Robots

Other types of robotic platforms integrate filtration and UV-C irradiation in a compact design, aiming to combine advantages of both. The Autonomous Indoor Cleaning Robot (AICR), for instance, uses an impeller fan to draw ambient air through a HEPA filter while continuously exposing the filter to internal UV-C lamps [22]. In this configuration, UV-C is not directed into the room but onto the filter medium, inactivating trapped pathogens and potentially extending filter life. The filtered, disinfected air is then discharged back into the environment. The AICR platform (Figure 2.2) couples this air-cleaning module with autonomous navigation (based on VSLAM) to patrol small shops and office spaces [22].

Compared to stand-alone UV-C robots, such designs can operate more safely in environments with human presence, as UV-C radiation is confined within the device. Compared to purely mechanical filtration, the added UV-C disinfection of the filter may reduce viable pathogen load on the filter and lower the risk of re-emission. However, the overall air-cleaning effectiveness still depends on airflow rates, room mixing, and correct positioning or motion planning, and the system does not directly disinfect room surfaces.



Figure 2.2: Autonomous Indoor Cleaning Robot (AICR).

Smart and Sensor-Driven Air Purifiers

A final category encompasses “smart” purifiers and cleaning robots that integrate air-quality sensing, network connectivity, and control algorithms. A smart air purifier designed for small public spaces, for example, combines HEPA filtration with low-noise fans and embedded particulate or gas sensors to maintain pollutant levels below specified thresholds in cafés or waiting rooms [23]. Another system, developed as an air-pollution purifier, uses directional airflow and high-efficiency filters to create localized clean zones in indoor environments with poor general ventilation [24]. These devices often provide IoT connectivity for monitoring and remote control, and may implement rule-based or simple feedback control strategies based on sensor readings [23, 24].

This kind of devices also find their application at a domestic scale, an AI-powered cleaning robot prototype integrates floor-cleaning capabilities with gas sensors and an IoT platform (*NodeMCU* and *ThingSpeak*) to monitor indoor pollution and adjust cleaning behavior accordingly [25]. While its primary functionality is vacuuming and mopping, the system demonstrates how environmental sensing and cloud-based analytics (e.g., ARIMA forecasting of gas sensor data) can be coupled with a mobile platform in residential settings [25]. These smart purifiers and sensor-equipped robots illustrate a trend towards continuous, sensor-driven sanitization strategies that are inherently compatible with occupied spaces. Their advantages include the ability to adapt operation to measured air-quality conditions, provide feedback to users, and integrate with building management or smart-home systems. Limitations include the complexity of correctly interpreting IAQ metrics (e.g., relating CO₂ or VOC sensor readings to infection risk), potential calibration and reliability issues with low-cost sensors, and, in many current prototypes, relatively simple navigation and control architectures.

In summary, sanitization techniques in service robotics range from high-intensity, episodic disinfection (UV-C and hybrid UV-C/chemical robots operating in empty rooms) to continuous, occupancy-compatible air cleaning (HEPA-based purifiers and integrated HEPA+UV-C devices), and increasingly to intelligent, sensor-driven platforms that couple purification with environmental monitoring.

2.2 Environmental Sensing

Indoor Air Quality (IAQ) sensing provides the quantitative basis to evaluate ventilation adequacy, pollutant exposure, and—indirectly—the risk of airborne infection in enclosed spaces. Traditional approaches rely on fixed sensor networks or “nearable” devices permanently installed in representative locations of a building, typically monitoring carbon dioxide (CO₂), temperature, relative humidity, particulate matter (PM), and volatile organic compounds (VOCs) [7, 26]. However, the cost and complexity of installing dense networks of calibrated sensors in existing buildings, together with the strong spatial heterogeneity of IAQ, have motivated a growing interest in mobile sensing using service robots equipped with environmental sensors [27, 28]. More recently, hybrid architectures combining fixed sensing infrastructure and mobile robots have been proposed to exploit the complementary strengths of both modalities [29].

Fixed and nearable IAQ sensing

Fixed IAQ monitoring solutions generally adopt either centralized building automation and control systems (BACS) with hard-wired sensors, or distributed wireless sensor networks (WSNs) and Internet-of-Things (IoT) nodes [30, 7]. For example, Corgnati et al. propose a nearable wireless device for indoor environments that continuously measures CO₂ temperature, humidity, and illuminance, and can trigger local actuators (e.g., window opening, lighting control) when predefined thresholds are exceeded [26]. Similar IoT-based platforms deploy low-cost CO₂ loggers in classrooms or offices, compute spatial or temporal averages over defined zones, and raise alarms when average concentrations remain above thresholds associated with increased health risks or degraded cognitive performance [27, 30].

These systems provide high temporal resolution at their installation points and can be tightly integrated with HVAC control. However, their spatial resolution is limited by sensor density: even in relatively simple spaces, large vertical and horizontal gradients in CO₂ and other pollutants have been observed, leading to “measurement blind spots” where conditions significantly differ from sensor readings [31, 29]. Retrofitting existing buildings with a sufficiently dense sensor network is often prohibitively expensive and logistically complex, particularly when hundreds of sensors would be required to cover all occupied zones [32, 33]. These limitations motivate the use of mobile service robots as flexible, repositionable sensing platforms able to sample many locations with a few sensors.

Mobile service robots for IAQ monitoring

A first family of works focuses on mobile robots whose primary function is environmental monitoring. Jin et al. introduce an “automated mobile sensing” paradigm, using a TurtleBot 2 platform carrying an environmental sensing module that measures CO₂, VOCs, PM_{2.5}, temperature, humidity, and light.[28] The robot, programmed under ROS, autonomously navigates an environmental chamber while a spatio-temporal interpolation algorithm reconstructs high-granularity fields of tracer-gas concentration and air-change effectiveness (ACE) from sparse mobile samples [28]. When benchmarked against a dense static network of ten CO₂ sensors installed according to ASHRAE 129, the mobile system reproduces local ACE values with low root-mean-square error and high correlation, demonstrating that a single navigation-capable platform can emulate the diagnostic capability of a much larger fixed network at a fraction of the sensor cost.

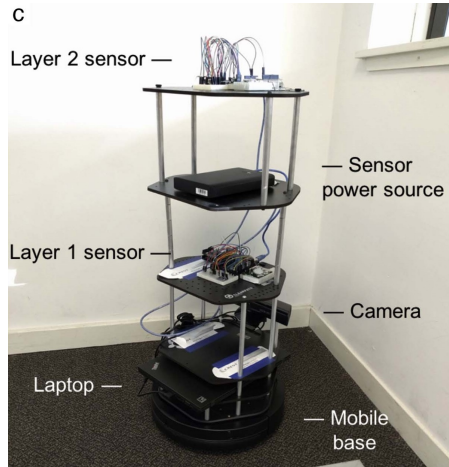
Yang et al. propose a similar mobile CO₂ sensing approach oriented to contaminant source localization [34]. Their omnidirectional robot, equipped with a 2D LiDAR and an NDIR CO₂ sensor, builds a 2D map of a laboratory using Rao-Blackwellised particle-filter SLAM, then follows a predefined trajectory to acquire CO₂ measurements under controlled window-opening and source-emission conditions. Using Kalman filtering and nearest-neighbor spatial interpolation to generate heat maps, they compare a network of nine fixed CO₂ sensors against the mobile platform. While static sensing offers continuity in time but basic spatial coverage, the mobile robot attains higher spatial sensitivity: the estimated position of the CO₂ source is within approximately 1.8 m of the ground truth, versus about 3.1 m for the fixed network [34]. The study explicitly highlights that mobile sensing captures stronger spatial gradients and is more sensitive to localized plumes, at the cost of sampling only one location at a time.

The Indoor Air Quality Detection Robot model by Simamora et al. further illustrates the design of a low-cost mobile platform dedicated to IAQ monitoring [35]. Their robot integrates a suite of gas and environmental sensors (SGP30 for CO₂ equivalents and VOCs, MQ-2 for smoke, DHT11 for temperature and humidity) on an ESP32-based chassis that performs room mapping using a time-of-flight distance sensor and simple regression-based wall reconstruction. IAQ is classified into discrete quality levels via Mamdani fuzzy logic, fusing multiple sensor inputs into a single health-relevant index [35]. Experimental evaluation in a test room reports average measurement errors around 10 % for gas sensors and mapping errors below 6 %, indicating that relatively inexpensive components can suffice for qualitative IAQ assessment when combined with appropriate classification logic.

Beyond pure monitoring, several works explore robots that both sense IAQ and

perform corrective actions. Abdallah and Nasr present a wheeled mobile manipulator equipped with CO₂, PM_{2.5}, VOC, temperature, and humidity sensors, which classifies IAQ into discrete “zones” based on WHO and European guideline values [36]. Depending on the zone, the system either recommends occupant actions (e.g., open windows, reduce occupancy) via a mobile application or autonomously sanitises the room by spraying a disinfectant solution using a gripper-mounted nozzle. Validation in university lecture halls shows that the robot can repeatedly survey the space, inform users of IAQ deviations, and execute sanitisation routines when concentrations of CO₂, VOCs and PM_{2.5} exceed acceptable ranges [36].

Palacín et al. analyze a humanoid-like omnidirectional service robot used as a mobile CO₂ measurement platform during the COVID-19 pandemic [32]. The robot, originally designed as an assistive platform, is retrofitted with an NDIR CO₂ sensor and deployed to patrol a multi-storey educational building, visiting classrooms according to a schedule derived from the occupancy timetable. Using a 2D LiDAR-based SLAM map, A* path planning, and waypoint-based missions, the robot samples CO₂ in each room for several minutes and reports the readings together with occupancy and exposure time. The retrospective analysis shows that the robot-based system can provide interpretable information on ventilation performance and infection risk when compared against CO₂-based risk models such as those of Di Gilio et al. and Bazant and Bush, while avoiding the need to install roughly ninety-five fixed sensors that would have been required to instrument all shared spaces in the same building [32].



(a) Robotic platform developed in [28].



(b) Humanoid-like omnidirectional robot detailed in [32].

Figure 2.3: Example of monitoring robotic platforms.

Assistive and social robots with IAQ sensing

Some mobile platforms integrate IAQ sensing into broader ambient-assisted living and human-robot interaction (HRI) scenarios. Marques et al. propose iAirBot, an assistive robot based on TurtleBot 2 that interfaces with a WSN of IAQ sensor nodes (iAirNodes) deployed in the environment [27]. Sensor nodes measure CO₂, VOCs and thermal conditions and transmit data via IEEE 802.15.4 to a gateway mounted on the robot, which in turn aggregates and uploads IAQ information to cloud services and to a Facebook page used by caregivers [27]. When poor IAQ is detected, the robot navigates towards the user using a Kinect-based localisation pipeline and informs them via speech synthesis, while simultaneously posting alerts on social networks. In this architecture the robot is both a mobile sink for the sensor network and an embodied interface for IAQ-related feedback in enhanced living environments.

A conceptually related approach is presented by Ribino et al., who employ a humanoid social robot in combination with fixed environmental sensors to monitor comfort parameters and query occupants about their perceived well-being [37]. While the emphasis is on subjective comfort rather than infection risk, such works illustrate the potential of mobile service robots to mediate between technical IAQ measurements and user-centric interventions, for example by recommending window opening, occupancy reduction, or schedule changes based on sensed and reported conditions.

Hybrid fixed-mobile architectures

Hybrid systems explicitly integrate fixed IAQ sensors with mobile robots to mitigate the weaknesses of each sensing modality. A recent example is the indoor CO₂ monitoring system with an integrated fixed sensor network and a mobile measuring robot (MMR) proposed by Tamura et al. for office environments [29]. Their architecture deploys ceiling-mounted CO₂ sensors to provide continuous coarse-grained monitoring, while a Cuboid-based MMR equipped with a CO₂ sensor, LiDAR, RGB and depth cameras, and speakers performs targeted, high-precision measurements on demand. When the cloud backend detects that the CO₂ level at any fixed sensor exceeds a configurable threshold, it dispatches the robot to a predefined measurement point near that sensor [29].

At each measurement point, the robot uses YOLOv3-tiny running on an NVIDIA Jetson module to detect the presence of occupants and decides whether a detailed measurement is required [29]. If people are present and CO₂ remains high, the system estimates future CO₂ evolution using feedforward and recurrent neural networks trained on historical sensor data, and issues voice prompts suggesting immediate corrective actions (e.g., opening windows, adjusting local ventilation).

Experiments in real offices show that differences between co-located fixed and robot-mounted CO₂ sensors remain within the accuracy bounds of the sensors (on the order of tens of ppm), while the hybrid strategy reduces measurement time and improves the detection of spatial gradients and blind spots that would be missed by ceiling sensors alone [29]. This demonstrates that integrating fixed and mobile sensing can support occupant-centric IEQ management even in buildings lacking fine-grained BACS.

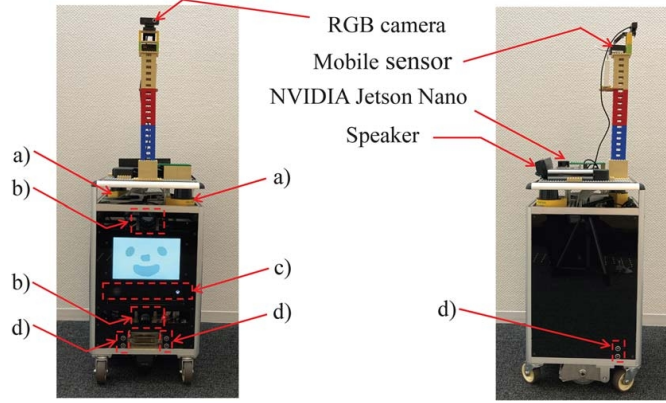


Figure 2.4: Mobile Measuring Robot detailed in [29].

Earlier works already hinted at this hybrid paradigm. Jin et al. note that their automated mobile sensing platform can either operate alone or in combination with existing static networks, leveraging mobile data to densify spatial coverage and using static nodes for continuous background monitoring [28]. Palacín et al. similarly discuss the potential to couple their mobile CO₂ measurements with demand-controlled ventilation systems, although in their deployment the robot was not yet integrated with the building’s HVAC controls [32]. These studies collectively suggest that fixed sensors are well suited for temporally continuous, low-maintenance monitoring at a limited number of strategic points, whereas mobile robots can be invoked when more detailed spatial diagnosis, local source localization, or direct interaction with occupants is required.

Comparison and implications for this work

From the surveyed literature, three main patterns emerge regarding environmental sensing technologies for mobile service robots in IAQ applications:

- *Fixed-only IoT and nearable systems* provide continuous monitoring and straightforward integration with building control, but suffer from limited spatial resolution and high infrastructure cost when dense coverage is needed [26, 27, 31].
- *Purely mobile sensing platforms* can achieve high spatial granularity and flexible deployment with a small number of sensors; they are particularly effective for tracer-gas experiments, pollutant source localization, and patrol-based CO₂ surveillance during events such as the COVID-19 pandemic [28, 34, 32, 35]. Their main limitations are the sequential nature of measurements, susceptibility to sensor response delays during motion, and the need for robust navigation and mapping.
- *Hybrid fixed-mobile architectures* attempt to combine the strengths of both, using static sensors for baseline monitoring and mobile robots for demand-based, occupant-centric measurements and interventions [29, 27]. Early results indicate that such systems can reduce blind spots, shorten measurement times, and support advanced functions such as predictive IEQ control and multimodal feedback (voice, social networks).

This thesis builds on these insights by adopting mobile sensing as the primary mechanism for IAQ monitoring, with CO₂ used as a proxy for ventilation adequacy and airborne infection risk, and by considering how a mobile service robot can be integrated into broader IAQ management strategies. In contrast to many proof-of-concept platforms based on compact research robots with limited payload and autonomy,[28, 34, 35] the proposed system targets a heavy-duty, ROS 2-based platform capable of carrying both environmental sensors and air-filtration hardware. The literature on fixed, mobile, and hybrid sensing reviewed in this section provides the foundation for the CO₂-driven navigation and control strategies explored in the subsequent chapters.

2.3 Map-based navigation

Map-based navigation facilitates path-planning and localization in known environments, enabling service robots to perform tasks such as patrolling, monitoring, and cleaning with spatial awareness, but it may not always be possible to know the environment in advance, or to maintain an accurate map over time due to changes in layout, furniture, or occupancy. However, when navigating inside unstructured environments such as offices, schools, or hospitals, a map is often needed to plan efficient routes and ensure coverage of key area. To achieve that, maps often undergo a post-processing step after their creation to remove noise and features that may compromise navigation. This is employed in [32], where a map was required to plan exploratory missions for CO₂ monitoring in a multi-story educational building. The map was obtained during a supervised exploration phase where the environment was thoroughly mapped, then manual map refining and definition of reference points was performed to define the exploratory missions. With this method, the map only needs to be created once and can be employed to carry out the monitoring tasks with other platforms, as long as the environment does not change significantly. The 2-D map is created by using a variant of the Iterative Close Point (ICP) algorithm. In [29], the map is built by manually operating the robot during an initial exploration, then waypoints are designated for the following monitoring missions.

Other researches employ SLAM-based (Simultaneous Localization and Mapping) navigation with cameras and LiDARs. Yang et al. [34] made use of the open source *GMapping* algorithm, along with A* global path planning and DWA local obstacle avoidance for the mobile platform described in the research, but stated that changes in either illumination or topology of the surrounding environment cannot be avoided and affect the accuracy of localization and mapping. *VSLAM* (Visual SLAM) is employed in [22] to let a mobile HEPA/UV-C disinfection platform, denominated *AICR* (Autonomous Indoor Cleaning Robot), navigate autonomously in small shops and office spaces, while targeting populated areas to sanitize. The design of such platform does not involve monitoring of IAQ parameters, nor the creation of a heatmap for the analysis or the simple visualization of CO₂ or pollutant distribution inside the unstructured environment. Furthermore, the exploration logic behind the sanitization missions is not directly deployed onto the platform, but rather remotely executed on a server (remote PC) running ROS. Such system premises the deployment of multiple AICRs, each managing a fraction of the environment, which is fairly limited (150 cm²), leading to a significant increase in the overall cost of the system. In [15], it is stated that the majority of UV robots use prior maps and target locations specified by a human operator to generate disinfection plans.

Adaptive Monte Carlo Localization

In this work, localization is performed using Adaptive Monte Carlo Localization (AMCL) as implemented in the Nav2 stack [38]. AMCL is a particle-filter-based algorithm that estimates the robot pose $x_t = (x, y, \theta)$ on a known 2D occupancy grid, using odometry as motion input and laser scans as measurements. It represents the belief $bel(x_t)$ as a set of weighted pose hypotheses (particles) and updates them at each time step according to the motion and sensor models [39].

Compared to a fixed-size Monte Carlo Localization scheme, AMCL adapts the number of particles online. When the belief is spread out or multi-modal (for example during global localization or after a large disturbance), AMCL maintains more particles to cover multiple plausible poses. When the robot is well localized and the posterior is sharply concentrated, the algorithm reduces the particle set, lowering computational cost while preserving accuracy. In Nav2, this adaptation is based on KLD-sampling, which automatically increases or decreases the sample size to keep the sample-based belief within a user-specified error bound with high probability, without requiring a single, conservative particle count for all operating conditions [39].

In practice, AMCL enables reliable global localization and pose tracking on the static map used in this thesis, while keeping real-time performance on the Intel NUC. The configuration adopted here uses a laser-based sensor model and odometry-based motion model with parameters tuned for the office-like environments considered in 4; full probabilistic derivations of particle filtering and KLD-sampling are reported in A.

2.4 Behavior-based navigation

Behavior Trees

Behavior Trees (BTs) [40] represent a method to structure the decision-making process in autonomous systems, giving the opportunity to increase their complexity and making them both modular and reactive. BTs lay their origin in the computer game industry, where they were first employed to increase the modularity in the control structures of Non-Player Characters (NPCs). Up to that point, such control structures were typically implemented as Finite State Machines (FSMs), which consist of a set of states and transitions that are easy to understand and implement.

However, FSMs can become difficult to manage and scale as the complexity of the system increases, producing a tradeoff between reactivity and modularity. For a system to be reactive, there needs to be many transitions between its components, so the intervention on one or multiple components leads to mandatory re-evaluation of other system's components. BTs address this issue by structuring the decision-making process as a tree, with a parent-child relationship between nodes: internal nodes are called **control flow nodes**, while leaf nodes are called **execution nodes**. Control flow nodes determine the order of execution of their child nodes, while execution nodes perform specific actions or checks.

A BT begins its execution from the root node, which has no parent, sending signals at given frequencies, denominated as *ticks*, allowing the execution of a node. Once the children receives a tick, it immediately returns a status to its parent:

- **Success:** the node has completed its task successfully.
- **Failure:** the node has completed its task but failed to achieve its goal.
- **Running:** the node is still executing its task and has not yet completed.

It is possible to distinguish between four different types of control flow nodes, such as *Sequence*, *Fallback*, *Decorator*, and *Parallel*, and two types of execution nodes, *Action* and *Condition*.

- *Sequence*: the node routes the ticks to its children with a set order (typically, from left to right), until it finds a child returning either *Running* or *Failure*. This status is passed from Sequence node to its parent but does not make the control node tick the next child, if any. This behavior is instead executed if a child returns *Success*. If **all** children of the Sequence node return *Success*, the node itself transmits such status.
- *Fallback*: the node routes the ticks to its children and returns *Success* if a child returns *Success* or *Running*, and returns *Failure* if and only

if all children return *Failure*.

- *Parallel*: The Parallel node routes the ticks to all its children and returns *Success* if a certain number (M) of children, coming from a user-defined threshold, returns *Success*; it returns *Failure* if $N - M + 1$ children return *Failure*, where N is the number of children, with $M \leq N$.
- *Decorator*: the node manages the return status and the tick frequency of an only child according to some user-defined rule.
- *Action*: the node is a leaf node that executes an action when it receives a tick. If such action is completed, the node returns *Success*, if not, it returns *Failure*.
- *Condition*: the node checks a proposition upon receiving a tick and returns either *Success* or *Failure* depending on the validity of such proposition. *Running* status is not considered for this node.

Type	Symbol	Success	Failure	Running
Sequence	$\rightarrow$	upon Success of all children	upon Failure of a single child	upon Running of a single child
Fallback	$?$	upon Success of a single child	upon Failure of all children	upon Running of a single child
Parallel	$\Rightarrow$	upon Success of M children	upon Failure of $N - M + 1$ children	else
Decorator	$\diamond$	custom	custom	custom
Action	text	upon completion	upon failed completion	during completion
Condition	text	if True	if False	never

Table 2.1: Summary of both internal and leaf nodes.

Integrations

The theory behind BTs laid the foundations for a novel perspective of modeling robot behavior, replacing FSMs, but soon rose the need for practical, functional tools to implement such theory in robotic applications. In the last decade the most popular tool by far became *BehaviorTree.CPP*, a C++ library providing a framework to create BTs and custom nodes, which owns interesting and exclusive features such as the possibility to exploit asynchronous Actions and create trees that are created at runtime through an XML-based interpreted language.

The great flexibility and ease of use of this library made it a silent standard in the industry and proof of this is its integration with *ROS 2*, for which a repository of useful wrappers is currently maintained [41], and with the *Nav2* stack, which heavily adopts every aspect of the BT framework [38].

In this framework, nodes are designed to be conceptually close to functions, with the possibility of passing either parameters or arguments, as inputs, and extracting relevant information from them, in the form of outputs, while allowing communication between them. In *BT.CPP*, data flow management is assigned to ports which are linked together by a key/value storage system denominated as *Blackboard*. Input ports can read entries in the Blackboard, while Output ports can write into entries. More information about how *BT.CPP* library, including documentation

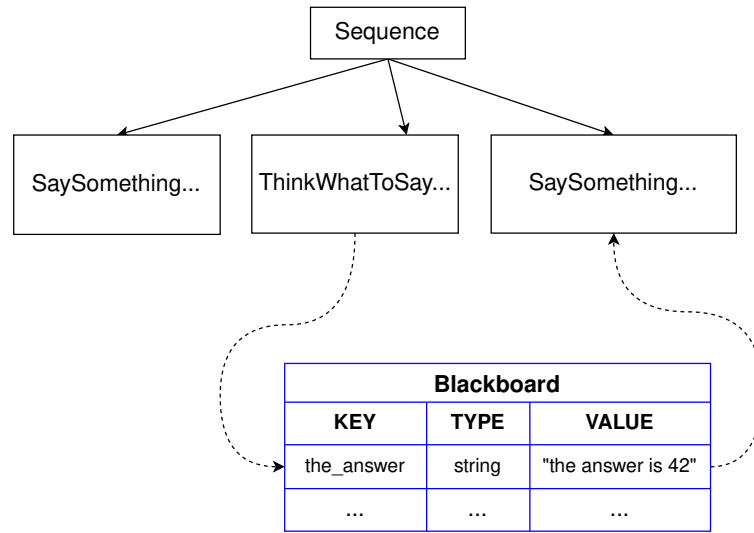


Figure 2.5: An example scheme of ports/blackboard interaction [42]

and tutorials, can be employed, is available at [42].

2.5 Robot Operating System 2

ROS 2 (Robot Operating System 2) is a middleware framework for developing distributed robotic applications. Unlike traditional robot control software, ROS 2 decomposes complex systems into loosely coupled *nodes* that communicate asynchronously through three main primitives: *topics*, *services*, and *actions* [43]. This publish-subscribe model, built on the Data Distribution Service (DDS) standard, enables scalability, real-time performance, and hardware abstraction essential for service robotics.

ROS 2 communication primitives are summarized in Table 2.2:

Table 2.2: ROS 2 Communication Primitives

Primitive	Pattern	Use Case	Example
Topic	One-to-many pub/sub	Continuous data streams	<code>/scan</code> , <code>/co2_level</code>
Service	Request- response	On-demand computation	<code>load_waypoint_list</code>
Action	Long-running + feedback + cancel	Complex oper- ations	<code>get_co2_value</code> , <code>navigate_to_pose</code>

In this work, the Nav2 stack provides `navigate_to_pose` actions for waypoint following, while custom Behavior Tree leaves use services (`load_waypoint_list`) and actions (`get_co2_value`, `update_heatmap`) to interact with the backend node `bt_server_node`. Quality of Service (QoS) profiles ensure reliable delivery of static maps ('TRANSIENT_LOCAL') and real-time sensor data, as described in Section 3.3.

Chapter 3

Methodology

This chapter describes the design and implementation process adopted to develop an autonomous CO₂-informed navigation and sanitization system for indoor environments. Starting from the physical integration of the robotic platform and its sensing peripherals, the work progresses toward the development of a ROS 2 Humble software stack that orchestrates autonomous waypoint navigation, real-time CO₂ data acquisition, and heatmap generation.

3.1 System Overview and Requirements

The system shall satisfy the following **Functional Requirements**, derived from the application domain and validated through simulation and real-world deployment:

Table 3.1: Functional System Requirements

Func. Req.	Description	Inputs	Outputs/Behavior
1	Autonomous way-point navigation	Waypoint pose list, Nav2 map	Robot reaches target ($\pm 0.3\text{m}$ tolerance)
2	CO ₂ measurement at waypoints	Waypoint index, number of samples	Averaged ppm value stored for waypoint
3	Dynamic way-point reordering	Previous CO ₂ measurements	Waypoints rotated: max-CO ₂ first
4	CO ₂ heatmap generation	Measurement list (x, y, v)	<code>co2_heatmap</code> , OccupancyGrid published
5	Mission management	Button press, battery state	Switch waypoint path or dock/shutdown

3.2 Robotic Platform

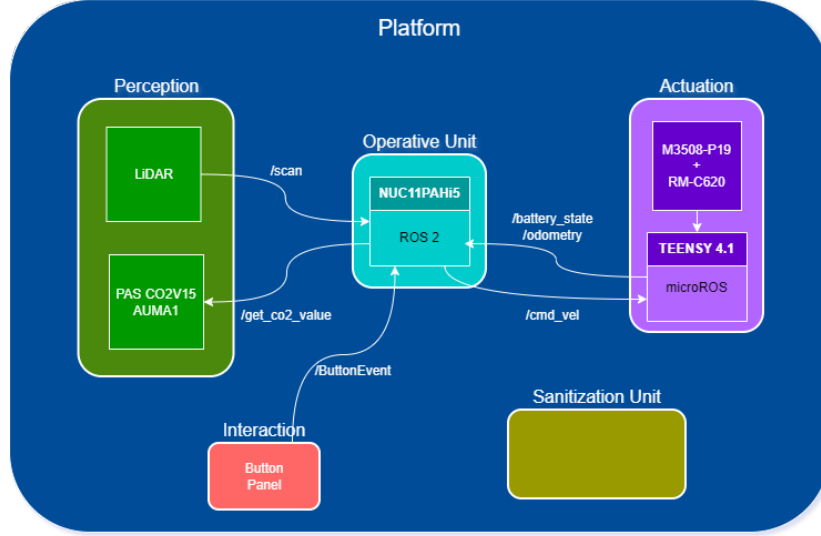


Figure 3.1: Robotic platform's diagram.

The robotic platform employed for the deployment of the system is composed of a structural timber slab designated for its mechanical strength properties with respect to its cost, in line with the concept of delivering an affordable and scalable platform. The base is supported by 4 pivoting wheels that enable the platform to sustain a maximum payload of 400 kg.

This solution is compliant with the employment of a broader type of sanitization units that can be integrated on the platform, from domestic to industrial grade. The platform also hosts two differential drive wheels under a coil spring suspension system, to enable free motion in every direction. This design perfectly fits the need for accurate maneuvers in an office environment, where narrow pathways and dynamical obstacles could impede the robot's motion. The suspension system has been specifically designed for the platform to ensure a smooth and reliable navigation even in non-uniform surface scenarios.

3.2.1 Sensors

The following paragraphs delve into the description of peripherals that allow the platform to interact, or even better, to read its surroundings.



Figure 3.2: Robotic platform designed for the project.

LiDAR

The RPLiDAR A1M8 [44] serves as the primary LiDAR sensor on the robotic platform, providing essential 360° environmental perception for navigation and SLAM tasks. This low-cost 2D laser scanner from SLAMTEC™ uses laser triangulation with a modulated infrared laser (Class I safety compliant), enabling reliable operation indoors and outdoors without direct sunlight by rejecting ambient light interference. It generates high-frequency point clouds via a spinning motor and adaptive speed control, outputting distance, angle, and quality data over UART for real-time processing in ROS2 environments. In the table below, a table containing the sampling data output available through the communication interface is presented.



Figure 3.3: RPLiDAR A1 mounted on the platform.

Table 3.2: RPLIDAR A1 Sample Point Data Information

Data Type	Unit	Description
Distance	mm	Current measured distance value between the rotating core of the RPLIDAR A1 and the sampling point
Heading	degree	Current heading angle of the measurement
Quality	level	Quality of the measurement
Start Flag	Boolean	Flag of a new scan

Separately integrated powered scanner and motor systems ensure precision, with self-diagnostic features that halt operation on laser faults or speed anomalies for safety and reliability. Its lightweight design and customizable mounting suit mobile robotics, supporting applications like obstacle avoidance and mapping. Further details about the employed sensor are displayed in the following table:

Table 3.3: RPLIDAR A1M8 Technical Specifications

Category	Item	Min	Typical	Max	Unit	Comments
Measurement Performance	Distance Range	—	0.15	12	m	White objects
	Angular Range	—	0–360	—	°	—
	Scan Field Flatness	−1.5	—	1.5	°	—
	Distance Resolution	—	0.5	—	mm	1% of distance (1.5 m range)
	Angular Resolution	—	1	—	°	5.5 Hz scan rate
	Sample Duration	—	0.125	—	ms	—
	Sample Frequency	—	8000	8010	Hz	—
	Scan Rate	1	5.5	10	Hz	360 samples/scan typical
Laser Power	Wavelength	775	785	795	nm	Infrared
	Power	—	3	5	mW	Peak power
	Pulse Length	—	110	300	μs	—
Misc	Weight	—	170	—	g	—
	Temperature Range	0	20	40	°C	Operating

CO₂ sensor

The sensor employed in this work is the **PAS CO₂V15 AUMA1** [45] and exploits a *Non-Dispersive Infrared* (NDIR) technology for CO₂ detection. The working principle of the sensor is the *photoacoustic spectroscopy* (PAS): when CO₂ molecules absorb infrared light, they generate a small change in the pressure. This variation is detected by an acoustic detector in the proximity of the sensor’s cavity, and a very accurate CO₂ concentration ppm reading can be delivered by the integrated microcontroller. This technique is favored over chemical CO₂-sensing and widely applied in the context of IAQ monitoring, where its accuracy has been deemed

valuable [46]. Furthermore, its limited size boosts its integrability to an even wider range of applications, while keeping a competitive cost.

For this particular integration, the sensor was encased in a 3D-printed prismatic hull to protect it from potential damage, and to contain as well an *ArduinoNanoTM* that exploits an existing I2C library to make use of CO₂ sensor's I2C ports and transfer incoming serial data to a USB port. This solution is justified by the Operative Unit's lack of direct serial interface, as most, if not every, commercial computer. By adopting this connection strategy, it was also possible to power such sensor with the same cable, being 5 V a sufficient supply.

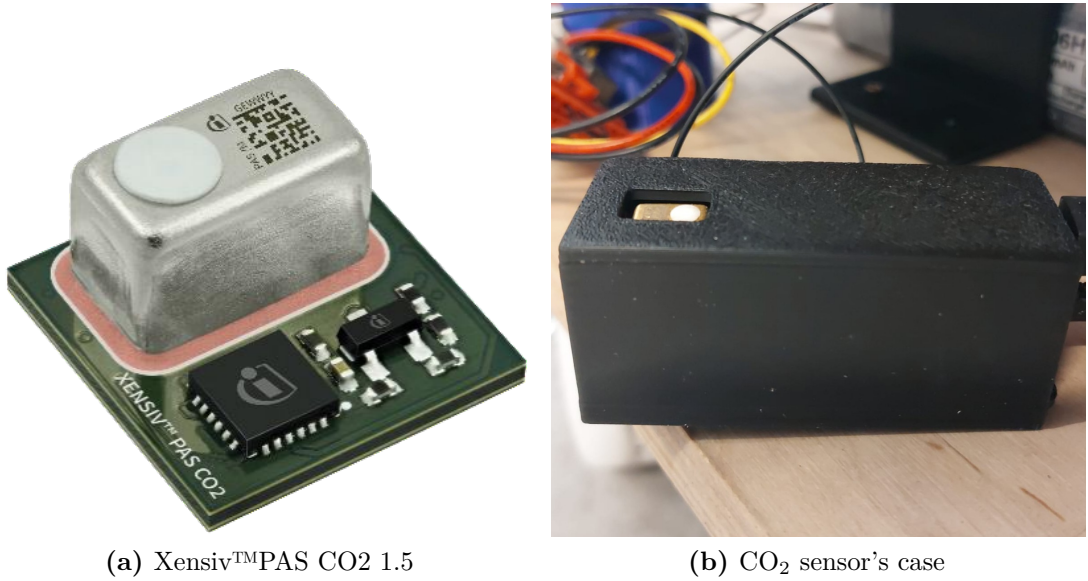


Figure 3.4: CO₂ sensor (Xensiv PAS CO2 1.5) and its protective case.

Table 3.4: Operating range

Parameter	Symbol	Values			Unit	Note/Test cond.
		Min	Typ.	Max		
CO ₂ measurement range	C_{CO2}	0	–	32000	ppm	functional measurement range
Ambient temp.	T_{amb}	0	–	50	°C	
Rel. hum.	rH	0	–	85	%	non-cond.
Pressure	p	750	1013	1150	hPa	–
Supply voltage	VDD3.3	3	3.3	3.6	V	–
	VDD5.0	4.45	5	5.6	V	–
Lifetime	T_{life}	–	10	–	Year	depends on mission profile

3.2.2 Actuators

The propulsion system relies on *DJI Robomaster M3508-P19* [47] brushless DC geared motor paired with the **RM-C620** [48] electronic speed controller, forming a compact drive unit suitable for mobile robotics. The M3508 integrates a built-in position (encoder) sensor for position feedback and an internal temperature sensor, enabling accurate monitoring and safer closed-loop operation. With an approximately 19:1 gear ratio from its planetary reducer, the motor can deliver the torque needed for traction while keeping speed in a range that is convenient for control on small to medium platforms.



Figure 3.5: Representation of *Robomaster*’s brushless motor.

Motor actuation is handled by the C620 controller, which uses a 32-bit driver and Field-Oriented Control (FOC) to achieve precise torque regulation when driving the M3508-P19. The C620 supports two input modes — PWM (50-500 Hz) and CAN bus — and, when using CAN, it can provide feedback such as motor temperature, rotor position, and rotational speed to the onboard computer. This industrial protocol is widely employed for its reliability, low latency, and robustness inside electromagnetically-noisy environments (e.g. offices). The CAN interface operates at 1 Mbps and allows multiple controllers on the same bus, which simplifies wiring and supports coordinated multi-motor platforms. In practice, this combination improves motion stability under load variations (e.g. uneven terrain or payload changes) and can help overall efficiency compared to simpler commutation schemes, supporting longer operating times.

Table 3.5: Characteristic parameters of the RoboMaster M3508 P19 motor

Rated Voltage	24 V
Torque Constant	0.3 N·m/A
Speed Constant	24.48 rpm/V
Speed/Torque Gradient	72 rpm/N·m
Mechanical Time Constant	49 ms
Phase Resistance	0.194 Ω
Phase Inductance	0.097 mH
Operating Temperature Range	0–50 °C
Max Permissible Winding Temperature	125 °C
Number of Pole Pairs	7
KN	210 N
Weight	365 g
Reduction Ratio	3591/187

Table 3.6: M3508 P19 performance parameters using the C620 speed controller at rated voltage

Rotational Speed (without payload)	482 rpm
Current (without payload)	0.78 A
Rated Rotational Speed	469 rpm
Rated Torque (max continuous torque)	3 N·m
Rated Current	10 A
Max Efficiency	70 %
Stall Torque	4.5 N·m
Stall Current	2.5 A

A *Teensy*TM 4.1 microcontroller serves as a low-level actuation interface between the central processing unit and the propulsion system, leveraging microROS for real-time bidirectional communication over ROS2 topics.

The central unit trasmits velocity commands via the `cmd_vel` topic, specifying the target linear velocity along the x -axis(v_x) and angular velocity about the z -axis (ω).

These kinematic references are converted by the microcontroller into differential wheel velocities and subsequently into torque commands for the C620 ESCs, which regulate the M3508-P19 motors through FOC.

In response, the *Teensy*TM publishes comprehensive proprioceptive feedback: motor telemetry acquired via CAN bus — including motor speeds, positions, torques, and temperatures — is processed to estimate robot odometry, providing position, orientation and linear/angular velocities derived from wheel kinematics. Concurrently, a TF transformation (`tf` topic) between `odom` and `base_footprint` frames

maintains spatial consistency within the ROS2 tf tree. This closed-loop exchange enables high-level navigation while delegating low-level stability and safety to the microcontroller.

3.2.3 Interfaces

The platform is also equipped with a button panel, shown in Figure 3.6, which lets the operator access to a series of buttons to instruct the system to follow a specific mission. At this state of development, three buttons in total are configured to provide the following instructions:

- **Button 0:** follow Mission 0
- **Button 1:** follow Mission 1
- **Button 2:** return to the docking position

The peripheral is able to interact through a direct USB connection to the operative unit. Key press registration is further discussed in Section 3.3.



Figure 3.6: Example of button panel peripheral used for the platform.

3.2.4 Sanitization Unit

The sanitization module integrated on the robotic platform is the **VFA Solutions ASPRA Lite S400** [49], a mobile air-purification unit designed to remove viruses, bacteria, fungi, and particulate matter from indoor environments, while also reducing gases and odors thanks to an integrated activated carbon filter. The device provides a simple on/off operation and delivers a nominal air-cleaning capacity of 400 m³/h with low power consumption (20 W) and a sound level of about 39 dB, making it suitable for continuous use in occupied spaces such as hospitals, offices, care homes. In this work, the ASPRA Lite S400 is mechanically mounted onto the robot and powered by a portable battery system capable of providing AC supply, ensuring autonomous operation without relying on a fixed mains connection.



Figure 3.7: ASPRA portable sanitizer mounted on the robot [49].

3.2.5 Operative Unit

The operative unit is the core of the development of this work as it coordinates every other peripheral connected to it, from sensors to actuators. Because of the platform's need to have a reliable, flexible, and autonomous operative unit, the choice fell on the *IntelTMNUC* series of mini PCs, which can be easily integrated into the platform, having a competitive, limited size. In particular, the model employed in this project is the **NUC11PAHi5**, whose technical specs are more than efficient to withstand the computational complexity of autonomous navigation algorithms, sensor fusion, and real-time data processing. A table containing all the characteristics of the NUC is provided below. The operative system running on

this unit is *Ubuntu 22.04*. The entire Ubuntu distribution is optimized to work with *ROS 2* [43] and its related release, which is *Humble Hawksbill* (*humble* for short). *ROS 2 Humble Hawksbill* is the eighth stable release for ROS 2 with Long Term Support (LTS), reaching its official End-Of-Life (EOL) in May 2027. In the years, it has been considered a standard choice for industrial applications.

3.3 Software Architecture

The core contribution of this thesis lies in the design and implementation of the software architecture that enables CO₂-informed navigation on the proposed platform. The architecture is built on ROS 2 Humble and the Nav2 stack and is organized around three main layers: (i) sensor integration and CO₂ data acquisition inside the backend node `bt_server_node`, (ii) behavior-tree-based mission control, and (iii) backend services for waypoint management and heatmap generation. The mission is structured as a cyclic, waypoint-based patrolling task with the following high-level behavior:

1. Load a list of predefined waypoints for the current mission.
2. For each waypoint: navigate to the target pose, acquire CO₂ measurements, and update the heatmap.
3. After completing a full cycle, reorder the waypoints so that the next mission starts from the location with the highest CO₂ concentration.
4. Repeat the cycle until either the battery becomes low (triggering docking) or the user terminate or switches the mission via the interface.

This behavior is implemented through a combination of ROS 2 nodes and Behavior Trees defined in the `co2_bt_navigation` package, which contains:

- XML description of the mission Behavior Trees.
- C++ plugins implementing custom BT action and condition nodes.
- Python nodes providing execution, supervision, and backend services.
- Launch files and map files used by Nav2 and the navigation pipeline.

3.3.1 Sensor integration and CO₂ measurement

The first step towards sensor-informed navigation was to make CO₂ readings available within the ROS 2 ecosystem. During early prototyping, this was achieved with a standalone node `co2_serial_node`, implemented in Python, that opened the serial port at 9,600 baud, read ppm values from the sensor, validated them as integers, and published them on a topic `/co2_level` using `std_msgs/msg/Int16`.

This design was convenient for debugging and sensor bring-up, as it decoupled hardware access from the rest of the system. In the final architecture, however, this functionality is integrated directly into the application backend node `bt_server_node`. Instead of exposing a raw `/co2_level` topic, `bt_server_node` embeds a `C02SensorSerial` helper class that manages the serial device internally

and provides a simple `get_value()` method. The CO₂ sampling action server `get_co2_value` (`co2nav_interfaces/action/SampleCO2`) uses this helper to:

- Acquire a configurable number of samples from the sensor (or from a simulated source in *sim* mode);
- publish feedback messages with each individual reading;
- compute the average CO₂ concentration over the requested samples;
- update internal statistics (minimum and maximum observed ppm) and associate the average value with the current waypoint index;
- return the averaged value to the Behavior Tree, which stores it on the blackboard for subsequent heatmap updates.

This refactoring removes the need for a dedicated CO₂ topic and keeps all sensor-specific logic confined inside `bt_server_node`, while the Behavior Tree nodes interact only through the high-level `SampleCO2` action interface.

3.3.2 CO₂-Based BT Navigation Package

The `co2_bt_navigation` package implements the mission logic for CO₂-informed waypoint navigation. It is a hybrid ROS 2 package (C++ and Python) that integrates tightly with Nav2's BT-based navigation framework. Conceptually, it provides:

- Mission Behavior Trees, expressed in XML.
- Custom BT nodes (plugins) for CO₂ sampling, heatmap update, and waypoint management.
- Python nodes for BT execution and mission supervision.
- A backend node that offers waypoint, CO₂ sampling, and heatmap services.

The package is organized as follows:

- `behavior_trees/`: XML files describing the mission trees (e.g., `WaypointsBattery` and `GoToDock`).
- `src/` and `include/`: C++ source and headers for custom BT plugins (actions and conditions).
- `co2_bt_navigation/`: Python nodes for BT execution and mission control.
- `launch/` launch files that start Nav2 together with the custom nodes.
- `maps/`: 2D occupancy grid maps used for localization and planning.

This separation allows mission logic to be modified by changing XML descriptions and parameters, without recompiling the behavior-tree execution engine.

Behavior-Tree Mission Design

In this work, Behavior Trees (BTs) are employed as the main mechanism to orchestrate navigation, CO₂ sampling, safety behaviors, and user-driven mission selection. Instead of implementing large finite-state machines, the mission is decomposed into modular BT nodes, which improve readability, extensibility, and reactivity to asynchronous events such as low battery or button presses.

The BT nodes implemented as C++ plugins fall into two main categories:

- **Condition Nodes**, which check system state (e.g., battery level, button events).
- **Action Nodes**, which trigger navigation, CO₂ sampling, heatmap update, waypoint loading, or system shutdown.

Below, the main custom nodes employed in the mission trees are summarized.

Condition Nodes

- **IsButtonExit**: returns **SUCCESS** when the user presses a different button than the one corresponding to the currently active mission. This enables mission interruption or switching at BT level.
- **IsBatteryLow**: subscribes to a `sensor_msgs/msg/BatteryState` topic and returns **SUCCESS** if the measured voltage falls below a configured threshold, signaling that docking should be initiated.

Action Nodes

- **SetGoalAction** (synchronous): iterates over a list of waypoint poses stored on the blackboard. On each tick, it outputs the next **PoseStamped** goal and increments an internal index; if the `reset_index` flag is set, it resets the iterator.
- **GetDockingGoal** (synchronous): constructs a docking goal **PoseStamped** from configuration parameters (x, y, qz, qw) expressed in the map frame and writes it to the blackboard.
- **ShutdownComputer** (synchronous): executes a system shutdown command to power off the onboard computer after safe docking.

- **GetCO2Measurement**: acts as a client for the **SampleCO2** action server provided by `bybt_server_node`. It requests a configurable number of samples, waits for completion, and writes the averaged CO₂ value to the blackboard.
- **UpdateHeatmap**: acts as a client for the **UpdateHeatmap** action. It sends the current goal pose and the corresponding averaged CO₂ value to the backend, which updates the internal heatmap and publishes an **Occupancy Grid**.
- **LoadWaypointList**: calls the `load_waypoint_list` service with the selected mission path index and writes the returned waypoint poses, their count, and the `reset_index` flag to the blackboard.

These nodes are combined into higher-level trees that encode the mission logic.

Mission Tree: **WaypointsBattery**

The **WaypointsBattery** tree implements the main CO₂ mapping mission. It cyclically visits all waypoints, collects CO₂ measurements, updates the heatmap, and remains continuously reactive to low-battery or user-interruption events.

The tree uses the BT blackboard to share data among nodes, including:

- `{@waypoint_path}`: integer selecting which waypoint list (e.g., 0 or 1) to use.
- `{@button_value}`: ID of the currently active mission button.
- `{poses}`: list of waypoints loaded from the backend.
- `{n_poses}`: number of waypoints in `{poses}`.
- `{reset_index}`: flag used by **SetGoalAction** to restart iteration at the beginning of a new round.
- `{goal}`: current waypoint pose (`geometry_msgs/msg/PoseStamped`).
- `{average}`: averaged CO₂ concentration associated with the current waypoint.

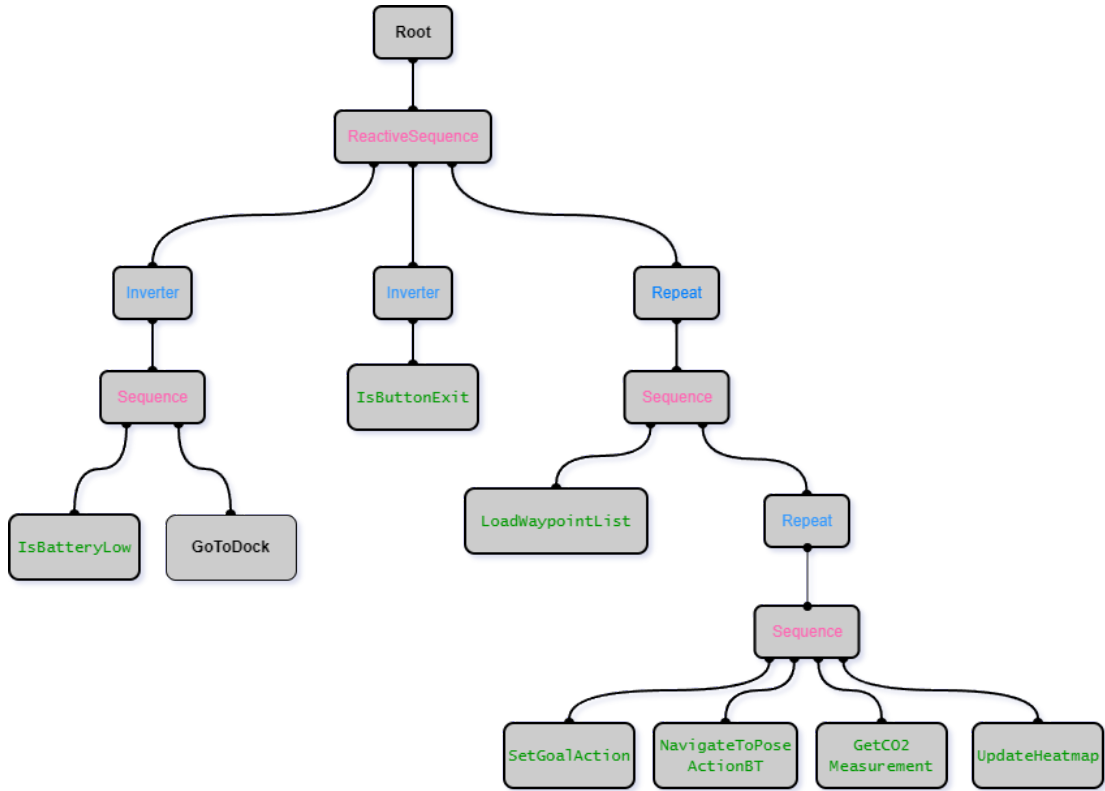


Figure 3.8: *WaypointsBattery* subtree.

At the root, *WaypointsBattery* uses a *ReactiveSequence* control node. This design ensures that safety and interruption conditions are re-evaluated at every tick, not only once at mission start. The tree is logically composed of three parts:

Battery guard with docking fallback If *IsBatteryLow* becomes SUCCESS, the tree branches into a *GoToDock* subtree that returns the robot to the docking pose and shuts down the computer. While the battery is above the threshold, this guard remains transparent and the mission proceeds.

Exit guard (button-based interruption) An inverted *IsButtonExit* node ensures that if the user presses a different mission button, the tree fails at the root, allowing the mission supervisor to cancel the running mission and return to idle.

Mission loop (rounds of waypoint exploration) A *Repeat* decorator executes the inner mission sequence for a fixed number of rounds (three in this implementation). Each round corresponds to a full traversal of the waypoint list,

potentially reordered based on CO₂ measurements from previous rounds. Within each round, the inner sequence performs:

1. **Round initialization** - LoadWaypointList

- Inputs: {@waypoint_path} and service name "load_waypoint_list".
- Outputs: {poses}, {n_poses}, {reset_index}. This node connects the BT to the waypoint's backend inbt_server_node.

2. Waypoint loop - iteration over {n_poses}

Each iteration consists of four steps:

- (a) **SetGoalAction**: selects the next waypoint pose from {poses} and writes it to {goal}.
- (b) **NavigateToPoseActionBT**: sends {goal} to the Nav2 `navigate_to_pose` action and waits for successful navigation.
- (c) **GetCO2Measurement**: requests CO₂ samples from the backend, computes the mean, and writes it to *average*.
- (d) **UpdateHeatmap**: sends {goal} and {average} to the `update_heatmap` action server, which updates the internal CO₂ map and publishes the latest heatmap.

This structure cleanly separates where the robot should go (waypoint list), how it moves (Nav2), and what application-level work is done at each waypoint (CO₂ sampling and heatmap update).

Docking Subtree: *GoToDock*

The `GoToDock` subtree implements the safe termination behavior when the battery is low or when the operator requests docking. It is organized as a *ReactiveSequence* with:

- An exit guard based on `IsButtonExit` (inverted), so that user commands can still interrupt the docking procedure if needed.
- A docking sequence that:
 1. Uses `GetDockingGoal` to fetch the docking pose (stored as a parameter and written to a global blackboard, e.g., {@docking_goal}).
 2. Invokes `NavigateToPoseActionBT` with this docking goal.
 3. Calls `ShutdownComputer` to power down the onboard PC once the robot has reached the docking station.

This subtree reuses the same design principles as the main mission tree: safety and operator control remain active at all times.

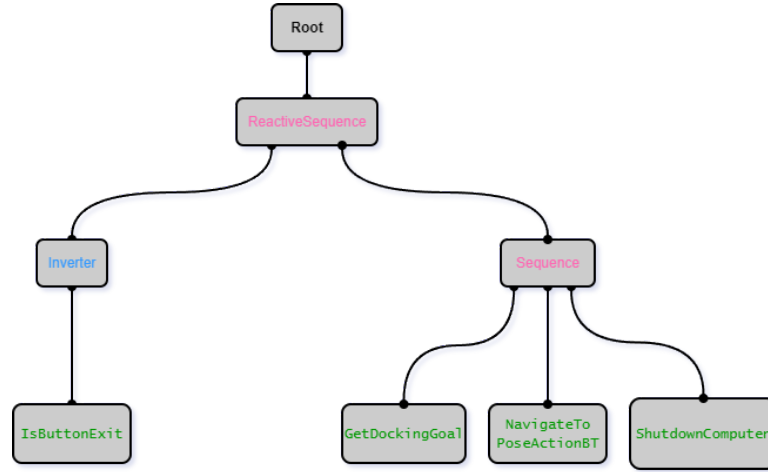


Figure 3.9: *GoToDock* subtree.

A complete depiction of the main tree is available in B.

3.3.3 Runtime Integration: Execution, Supervision, and Backend

The BTs described above are declarative specifications. At runtime, three main ROS 2 nodes provide execution, supervision, and backend functionality: `bt_executor`, `bt_nav_controller`, and `bt_server_node`. Plus, a python node handles key registration for a HMI.

3.3.4 Interface (`button_publisher`)

This Python node bridges an input device to the ROS2 ecosystem by publishing button events as custom `ButtonEvent` messages. It continuously monitors raw input events, filtering for keyboard presses. Mapped scan codes are converted to logical button IDs (`BUTTON0`, `BUTTON1`, `BUTTON2`) and published with `PRESSED/RELEASED` states.

The strength of this simple node lays in its single-threaded, non-blocking event loop, which is suitable for real-time robotics, and its device path, which is configurable from the script.

BT Executor (`bt_executor`)

The `bt_executor` node is implemented using the `BT::TreeExecutionServer` from the `behaviortree_ros2` framework and exposes an action server `/bt_execution`. When it receives a goal, it:

- Load the requested BT XML file (e.g., “WaypointsBattery”, “GoToDock”) from the package resources.
- Registers all custom BT plugins from `co2_bt_navigation`.
- Parses the JSON payload contained in the goal (e.g., `{"@button_value":0}, {"@waypoint_path":0}`) and writes the corresponding key-value pairs into the global blackboard.
- Starts ticking the tree until it reaches a terminal status (either `SUCCESS` or `FAILURE`) otherwise the goal is cancelled by a client.

This node decouples mission specification (in XML) from execution, and allows different missions to be launched dynamically at runtime.

Mission Supervisor (`bt_nav_controller`)

The `bt_nav_controller` node acts as a mission supervisor and human-machine interface. It subscribes to:

- `battery_state` for battery monitoring.
- `ButtonEvent` for operator input.

Internally, it maintains a simple state variable indicating whether a BT mission is running and which button is currently associated with it. Its main logic is:

- In **idle** state:
 - Wait for a button press.
 - On `Button0` press: send a `bt_execution` goal with:
`target_tree = "WaypointBattery",`
and payload `{"@button_value":0,"@waypoint_path":0}`
 - On `Button1` press: analogous, but with `@waypoint_path = 1`.
 - On `Button2` press or low battery: send `target_tree = "GoToDock"`
- During **mission execution**, if a different button is pressed, cancel the current `bt_execution` goal and return to idle.

In this way, `bt_nav_controller` provides a simple but effective bridge between the physical interface (buttons, battery) and the BT execution layer.

Application Backend (`bt_server_node`)

Finally, `bt_server_node` provides the application-level services and action servers required by the BT leaf nodes. It centralizes waypoint management, CO₂ sampling, and heatmap generation.

Waypoint management - `load_waypoint_list` service

- Loads a list of waypoints from YAML files based on the requested `waypoint_path`.
- Maintains a record of CO₂ measurements associated with each waypoint.
- Reorders the waypoint list so that the waypoint with the highest CO₂ value becomes the first waypoint in the next cycle, while preserving the relative order of the remaining waypoints (circular rotation).
- Returns the sorted list and its size to the BT.

CO₂ sampling - `get_co2_value` action

- Acquires a configurable number of samples from the CO₂ sensor (either from the real serial interface or from a simulated source).
- Publishes feedback with each sample to support monitoring.
- Computes the average concentration and stores it internally, associating it with the current waypoint index.
- Returns the average to the BT, which writes it into `{average}`.

Heatmap generation and interpolation - `update_heatmap` action

- Receives a pose and a CO₂ average, appends the measurement to an internal list, and updates a dictionary keyed by the spatial coordinates.
- Periodically generates a 2D grid that covers either the full map or the minimal bounding box around the measurements, at the same resolution as the occupancy grid.
- Fills cells coinciding with measurement locations with their measured CO₂ values.

- Uses an exponential, distance-weighted interpolation to estimate CO₂ concentration at unmeasured cells. The interpolated values are clamped to the maximum observed value for numerical robustness.
- Normalizes the resulting grid to a 0 – 100 scale and publishes it as an `OccupancyGrid` message on a `/co2_heatmap` topic.
- After a configurable number of new measurements, saves the current heatmap as a PNG image and exports raw and interpolated data as JSON for offline analysis.

The interpolation algorithm (Algorithm 1) computes each unmeasured cell c with center (x_c, y_c) :

$$S_c = \sum_{(x_m, y_m, v_m) \in \mathcal{M}} \exp(-\lambda \cdot d_m) \cdot v_m \quad (3.1)$$

where $d_m = \|(x_m, y_m) - (x_c, y_c)\|_2$ is the Euclidean distance to measurement m , and λ is the scaling factor controlling the decay rate. The final cell value is clamped as $H_c = \min(S_c, v_{\max})$, where $v_{\max}$ is the maximum observed CO₂ value. This exponential kernel $w(d) = \exp(-\lambda d)$ offers several advantages over classical *Inverse Distance Weighting* (IDW), which uses $w(d) = \left(\frac{1}{d}\right)^p$:

- **Smoothness:** Exponential decay produces continuous, differentiable gradients without the sharp fall-offs or oscillations common in high- p IDW.
- **Computational simplicity:** No division by zero handling or power computation required.
- **Interpretability:** The parameter λ directly controls the spatial influence radius (e.g., $\lambda = 0.6$ gives significant weight up to $\sim 3 - 5$ m).

Intuitively, the interpolation treats each recorder CO₂ measurement as a “source” that influences nearby cells in the grid, with an influence that decays exponentially with the squared Euclidean distance. For every grid cell, the algorithm accumulates contributions from all measurements, assigning higher weights to those that are spatially closer to that cell and lower weights to distant points. No weight normalization is applied, so region with multiple high-CO₂ measurements naturally appear as stronger “hotspots” in the final heatmap. The decay parameter λ directly controls the effective radius of influence: larger values produce more localized hotspots, while smaller values yield smoother, more diffuse maps.

However, the implementation has limitations that reflect its focus on qualitative visualization and real-time performance rather than quantitative accuracy:

- **No weight normalization:** S_c is a raw-weighted sum, not divided by $\sum w_m$, which can cause overestimation near clusters of high-CO₂ measurements.

- Fixed λ value is not adaptive to measurement density or map scale (it has to be tuned).
- Simple clamping: $\min(S_c, v_{\max})$ prevents extrapolation beyond observed values but ignores potential trends.

These choices are deliberate for the application context: the algorithm is real-time friendly ($\mathcal{O}(N \cdot M)$) with $N \leq 20$ measurements and $M \leq 50,000$ cells completes in $\sim 2\text{--}3$ s on the NUC, which is sufficient for periodic updates every 4 waypoints during navigation.

Algorithm 1 Exponential Distance-Weighted CO₂ Interpolation

```

1: Input:  $W, H, r, (x_0, y_0), \mathcal{M} = \{(x_m, y_m, v_m)\}, \lambda = 0.6, v_{\max}$ 
2: Output:  $H_{\text{grid}} \in \mathbb{R}^{H \times W}$ 
3:  $H_{\text{grid}}[i, j] \leftarrow \text{NaN} \ \forall i, j$ 
4: for  $(x_m, y_m, v_m) \in \mathcal{M}$  do
5:    $(i_m, j_m) \leftarrow \lfloor (x_m - x_0, y_m - y_0)/r \rfloor$ 
6:   if valid indices then  $H_{\text{grid}}[i_m, j_m] \leftarrow v_m$ 
7:   end if
8: end for
9: for  $i = 0$  to  $H - 1$  do
10:  for  $j = 0$  to  $W - 1$  do
11:    if  $\neg \text{isnan}(H_{\text{grid}}[i, j])$  then continue
12:    end if
13:     $(x_c, y_c) \leftarrow (x_0 + j \cdot r, y_0 + i \cdot r)$ 
14:     $S \leftarrow \sum_{(x_m, y_m, v_m) \in \mathcal{M}} v_m \cdot \exp(-\lambda \cdot \sqrt{(x_m - x_c)^2 + (y_m - y_c)^2})$ 
15:    if  $S > 0$  then  $H_{\text{grid}}[i, j] \leftarrow \min(S, v_{\max})$ 
16:    end if
17:  end for
18: end for

```

Overall, this architecture cleanly separates mission orchestration (BT XML), execution (`bt_executor`), supervision and HMI (`bt_nav_controller`), and application services (`bt_server_node`), all built on top of Nav2's navigation infrastructure.

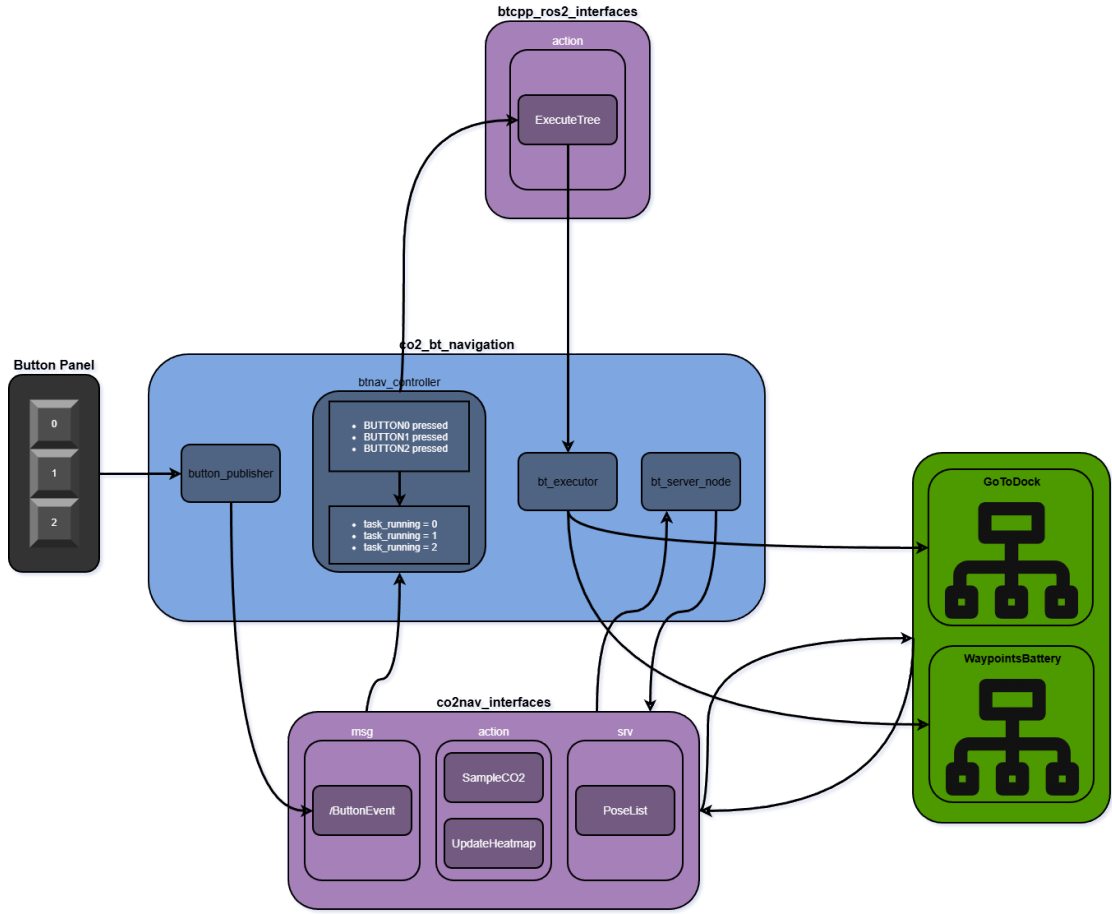


Figure 3.10: Overall system's structure and main interactions.

3.3.5 Composable strategy

A containerized microservice architecture based on Docker Compose was adopted for deploying the ROS 2 navigation stack, decomposing the system into dedicated services rather than a monolithic launch file. This approach leverages Docker Compose's declarative orchestration to manage dependencies, ensuring ordered startup while enabling independent scaling, updates, and simulation overrides.

Key services in `compose.yaml` include:

- **discovery_server**: Centralizes Fast DDS discovery for efficient ROS 2 networking.
- **bringup**: initializes robot platform, TF-tree, and low-level drivers.

- **lidar**: Handles RPLiDAR sensor interface and 2D perception pipeline.
- **navigation**: Runs Nav2 with environment-specific map and AMCL params, depending on **bringup** and **lidar**.
- **bt_executor**, **bt_server**, **btnav_controller** : execute high-level BTs for mission orchestration atop Nav2.

A system boot on the central unit, Compose automatically launches essential services, achieving full operational readiness without manual intervention —contrasting traditional manual launch sequences. This enhances maintainability, CI/CD integration, and real-to-sim portability.

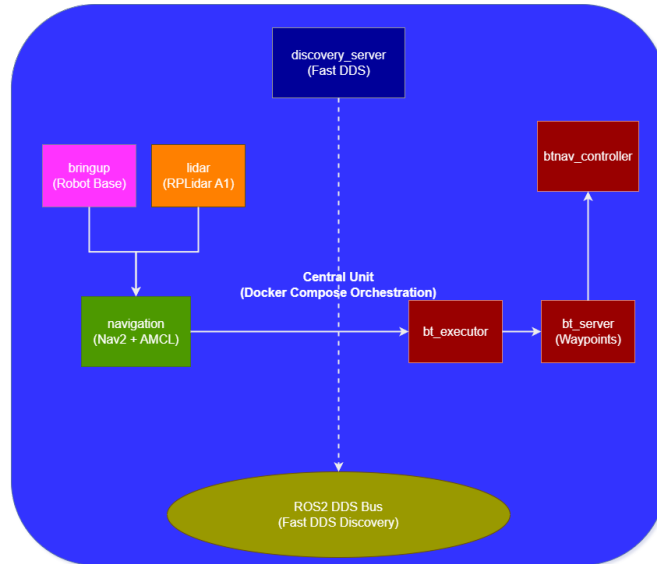


Figure 3.11: Compose structure.

Chapter 4

Experimental Evaluation

The experimental section of this thesis focuses on the validation of the developed system by evaluating the efficacy and the correct function of its navigation strategy. In the following sections, two ways of validation are explored. The first one tackles validation in a pure, simulated environment, while the second one, the most important one, tackles validation in a real deployment scenario.

4.1 Simulated Experiments - first approach

To conduct a proper evaluation of the system's capabilities in a simulated environment, the first task to approach is the creation of said simulated environment, so a world in *Gazebo Classic* was created to represent an appropriate environment for the service robot to interact with.

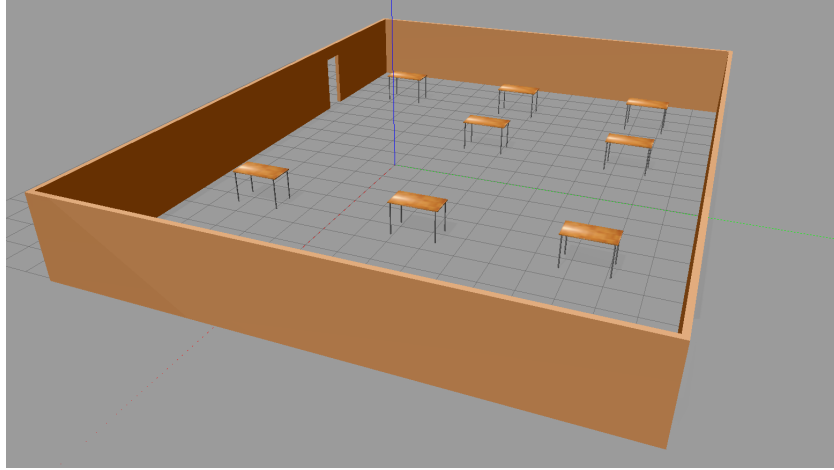
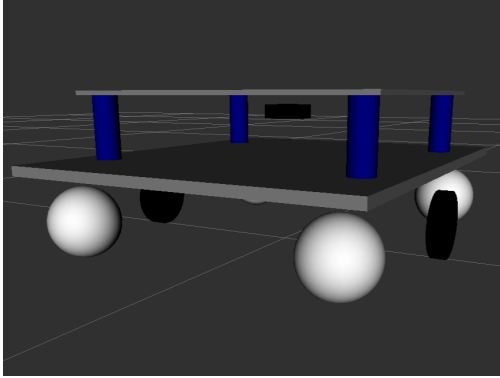
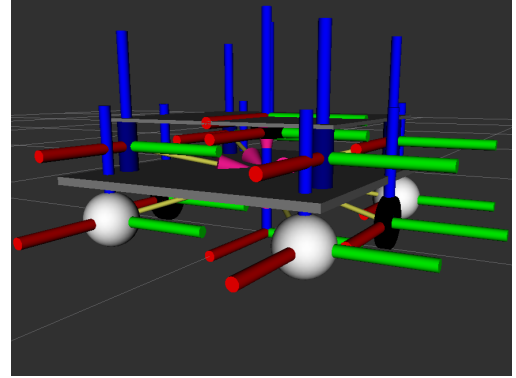


Figure 4.1: World created in *Gazebo* for the simulation.

The choice of including desks in the scene was made to imitate a work environment in the most simplistic way, with world features that may resemble what the actual robot could deal with during a real patrolling mission. Also, a URDF model of the robot, with its real dimensions, plus collisions and inertia, was created for the same reasons. The LiDAR was simulated as well, with position almost identical to where the true LiDAR is positioned on the actual platform. Pivoting wheels are modeled by approximating its shape to spherical because of similarities between their behaviors.



(a) Without TF



(b) With TF

Figure 4.2: URDF robot model.

First, the robot was manually guided through the simulated environment to produce a 2D scan of the map, shown below. It is possible to notice that such map is a

perfect representation of the Gazebo world.

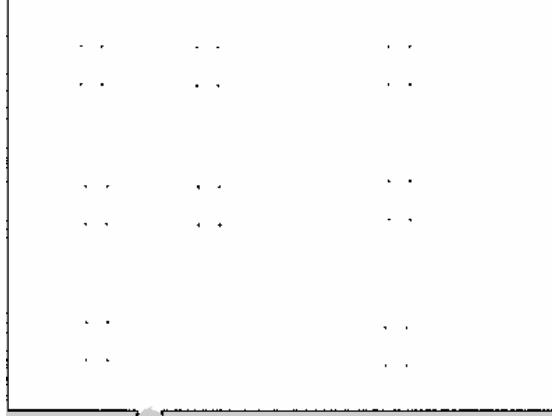


Figure 4.3: 2D map of the simulated office environment.

After the definition of a map, mandatory to assist the robot in its navigation, it was necessary to define a set of waypoints for CO₂ monitoring. 10 waypoints are selected in specific poses inside the environment for ease of conceptual explanation.

Waypoint	x coord.	y coord.
1	-9.00	-4.00
2	-5.00	2.00
3	-9.00	9.00
4	-3.00	6.00
5	9.00	9.00
6	4.00	5.00
7	9.00	1.00
8	9.00	-4.00
9	4.00	-1.00
10	-1.00	-4.00

Table 4.1: Waypoints coordinates

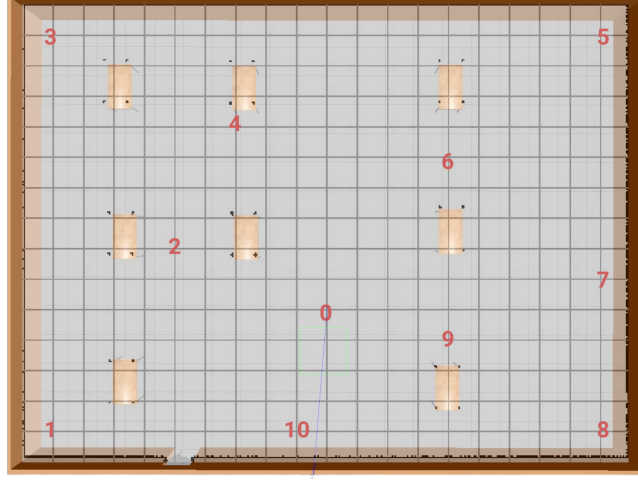


Figure 4.4: Visual representation of waypoints on the 2D map. "0" indicates the origin $[0,0]$.

To start the simulation, the composable strategy detailed in 3 was put aside in favor of a single launch file responsible for executing the majority of components to make the simulation work. The launch file first sets `ROS_DOMAIN_ID` to prevent conflict with the main containerized system, processes the robot URDF from a Xacro file, and launches *Gazebo* with the simulated environment while spawning the robot and running `robot_state_publisher` with simulation time. It then brings up the Nav2 stack via an external launch file, which is shared with the main application, using configurable arguments. Finally, it launches `bt_server_node`, `bt_executor`, and `btnav_controller` for high-level behavior tree navigation, and starts *RViz* after a short delay with a predefined configuration, so the whole simulated CO₂-informed navigation system is ready to test.

On launch, the robot is spawned at a parameter-defined origin pose. Through an executable capable of simulating system's stimuli (e.g. button presses, battery status change), mission 0 — detailed in 3.3 — is executed.

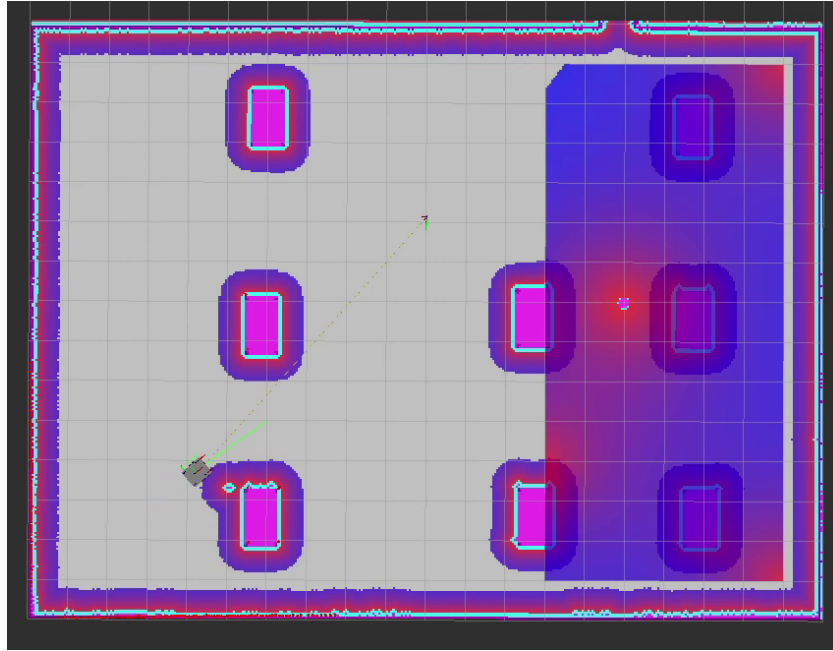


Figure 4.5: chopper platform navigating through the environment, approaching waypoint 6. Heatmap of the first 4 waypoints already created.

The robot completes its first mission cycle, collecting simulated CO₂ ppm values, available in the table below:

Waypoint	CO ₂ average reading
1	854.9 ppm
2	902.6 ppm
3	856.9 ppm
4	757.7 ppm
5	714.0 ppm
6	858.1 ppm
7	914.1 ppm
8	793.3 ppm
9	834.5 ppm
10	946.9 ppm

Table 4.2: Simulated CO₂-informed navigation, first cycle.

For the first mission cycle, *Waypoint 10* showed the highest average CO₂ reading among all waypoints. Upon completion, the sorting behavior activates. The new

waypoint list, related to the second cycle, has *Waypoint 10* as its first waypoint to explore, following through in a cyclic fashion: *waypoint 1* is the second waypoint to explore, until reaching *Waypoint 9*.

```
[bt_server_node-6] [INFO] [1772551812.224577283] [bt_server_node]: [LoadWaypointListServer] Beginning sorting
[bt_server_node-6] [INFO] [1772551812.224852284] [bt_server_node]: [LoadWaypointListServer] Sorting ended
[bt_server_node-6] [INFO] [1772551812.225221282] [bt_server_node]: [LoadWaypointListServer] Returning
[component_container_isolated-5] [INFO] [1772551812.241094007] [bt_navigator]: Begin navigating from current location (-0.92, -3.98) to (-1.00, -4.00)
```

Figure 4.6: `bt_server_node` sorting waypoint list and evaluating *Waypoint 10* as first waypoint to explore in second cycle.

Waypoint	CO ₂ average reading
10	779.0 ppm
1	746.1 ppm
2	911.9 ppm
3	814.5 ppm
4	887.2 ppm
5	875.0 ppm
6	795.6 ppm
7	777.1 ppm
8	814.8 ppm
9	816.5 ppm

Table 4.3: Simulated CO₂-informed navigation, second cycle.

Given the results of the second cycle of exploration, it is clear that *Waypoint 2* has the highest ppm value among all waypoints. Therefore, the waypoint is selected as the first waypoint to explore in the next cycle, as shown in the snapshot below.

```
[bt_server_node-6] [INFO] [1772552085.808330893] [bt_server_node]: [LoadWaypointListServer] Beginning sorting
[bt_server_node-6] [INFO] [1772552085.808837281] [bt_server_node]: [LoadWaypointListServer] Sorting ended
[bt_server_node-6] [INFO] [1772552085.809311215] [bt_server_node]: [LoadWaypointListServer] Returning
[component_container_isolated-5] [INFO] [1772552085.823888412] [bt_navigator]: Begin navigating from current location (4.19, -0.99) to (-5.00, 2.00)
```

Figure 4.7: `bt_server_node` sorting waypoint list and evaluating *Waypoint 2* as first waypoint to explore in third cycle.

The exploration then continues with its last cycle for which the results are irrelevant, since the robot is expected to shut down after its completion. During the simulation, the platform successfully avoided all obstacles while computing optimized path to

reach its targets. Other trials involved testing operational changes during mission, such as path switching or battery depletion, through a Python node able to simulate such behaviors and to communicate with `btnav_controller`. Upon receiving the aforementioned stimuli, the platform was able to rapidly change its goal either to reach the first position of a different set of waypoints (e.g., from `waypoint_path_0` to `waypoint_path_1`) or to return to dock (e.g., `Button2` selected, voltage falling under given threshold). This is sufficient to state that the system is capable of respecting the designed safety measures.

4.2 Simulated Experiments - second approach

A different approach was employed to test the ability of the system to detect CO₂-dense areas and orchestrate proper patrolling paths. To deliver this test, a simple python script was designed to create a publisher node capable of delivering a map organized in quadrants, each given a specific CO₂ level.

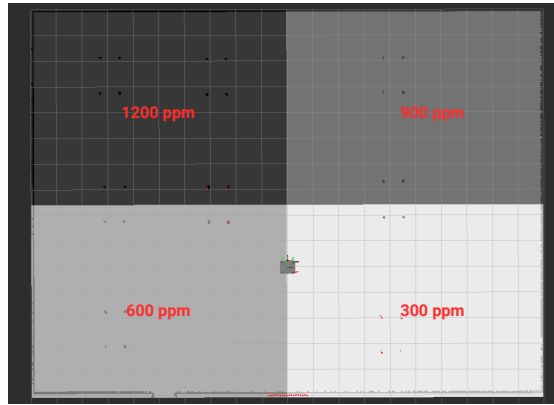


Figure 4.8: Simulation map divided in 4 quadrants. Every quadrant is associated with a CO₂ value (in ppm).

To carry out this test, it was decided to employ the same waypoint mission path employed in 4.4. As the terminal snippets suggest, the system successfully detected changes in CO₂ concentration along the different zones, recognizing the upper left quadrant (1200 ppm) as the container for waypoints to start the next cycle from. The simulation was interrupted after the simulation of the first heatmap containing the exploration of all waypoints.

From the picture below, correlation between quadrants' and waypoints' CO₂ level can be evidenced. The highest level detected corresponds to the most CO₂-dense quadrant and its waypoints. In all experiments, the displayed CO₂ heatmaps are generated with the exponential distance-weighted interpolation described in 3.3.4

```
[bt_server_node_sp-6] [INFO] [1772710267.551016406] [bt_server_node_sp]: [Co2Measuremen
tServer] Value: 300
[bt_server_node_sp-6] [INFO] [1772710267.629151566] [bt_server_node_sp]: [UpdateHeatmap
ActionServer] Added CO2 avg: 300.0 ppm at (9.00, 1.00)
```

(a) Successfully detected CO₂ level for SE quadrant

```
[bt_server_node_sp-6] [INFO] [1772710328.469194898] [bt_server_node_sp]: [Co2Measuremen
tServer] Value: 600
[bt_server_node_sp-6] [INFO] [1772710328.541473660] [bt_server_node_sp]: [UpdateHeatmap
ActionServer] Added CO2 avg: 600.0 ppm at (-1.00, -4.00)
```

(b) Successfully detected CO₂ level for SW quadrant

```
[bt_server_node_sp-6] [INFO] [1772710249.922287417] [bt_server_node_sp]: [Co2Measuremen
tServer] Value: 900
[bt_server_node_sp-6] [INFO] [1772710250.001497052] [bt_server_node_sp]: [UpdateHeatmap
ActionServer] Added CO2 avg: 900.0 ppm at (4.00, 5.00)
```

(c) Successfully detected CO₂ level for NE quadrant

```
[bt_server_node_sp-6] [INFO] [1772710191.283717038] [bt_server_node_sp]: [Co2Measuremen
tServer] Value: 1200
[bt_server_node_sp-6] [INFO] [1772710191.357718221] [bt_server_node_sp]: [UpdateHeatmap
ActionServer] Added CO2 avg: 1200.0 ppm at (-3.00, 6.00)
```

(d) Successfully detected CO₂ level for NW quadrant

Figure 4.9: Overall caption for the 4 stacked subfigures

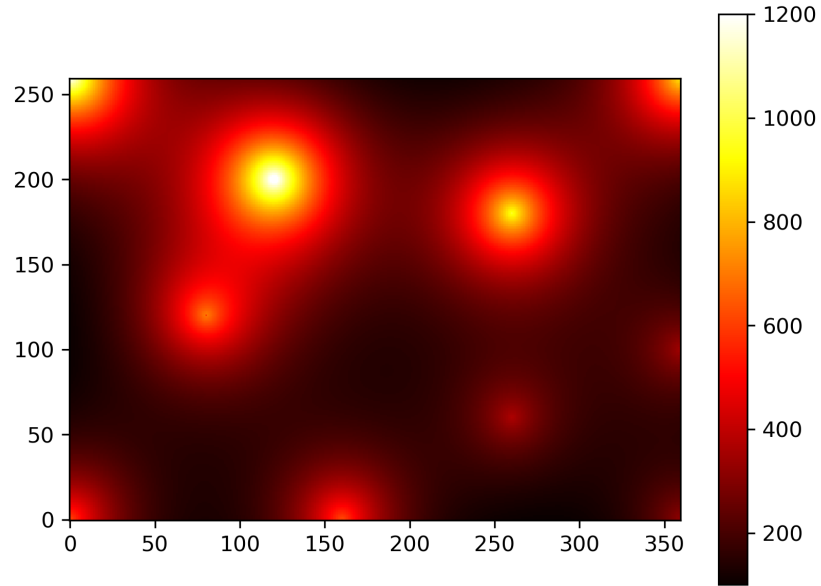


Figure 4.10: Heatmap processed by the server node after the exploration of the 12_{th} waypoint.

and should be interpreted as qualitative visualizations of relative high- and low-CO₂ regions rather than an absolute, metrically accurate concentration maps.

4.3 Real Deployment

Final evaluation of the system’s capabilities was carried out inside *Intesa San Paolo Innovation Center* (ISPIC) offices. Prior to the deployment, the environment was scanned to create a map as a reference for navigation



Figure 4.11: Map scan of ISPIC’s offices.

As previously described in the evaluation of the simulated environment, the definition of a waypoint path is key to a correct patrolling strategy. Based on the topology of the environment, shown in Figure 4.12, 14 waypoints are designated to create a coherent patrolling loop (*path 0*) that starts from *Waypoint 1*, which can also be treated as the platform’s dock, and evolves mainly along two corridors to avoid navigation through narrower passages, such as spaces between desks and drawers, that would make the patrolling path unoptimized and unnecessarily more fragmented. In the early stages of testing, another set of waypoint was considered for exploration and monitoring the lower map section (conference room). Due to structural changes which modified the topology of the environment to be explored, it was not possible to establish a new set of waypoints to carry out a satisfying validation test. This makes little difference to the demonstration of the system’s

capabilities of task completion, as analogous behavior and results could have been observed while choosing a different waypoint path.

Tests were conducted during work hours, where human presence is expected to increase CO₂ concentration near working stations. This setting was essential to evaluate the platform's capabilities of detecting "heated" areas of the work environment which to prioritize the sanitization process into. During the mission,

Waypoint	x coord.	y coord.
1	-2.12	-2.38
2	-2.99	-5.48
3	-4.84	-7.6
4	-7.86	-7.65
5	-13.88	-7.7
6	-19.79	-7.8
7	-23.79	-7.8
8	-27.94	-8.31
9	-24.68	-2.70
10	-21.1	-2.70
11	-18.54	-2.65
12	-15.26	-2.65
13	-13.03	-2.28
14	-6.89	-2.22

Table 4.4: Waypoints coordinates in ISPIC's offices.

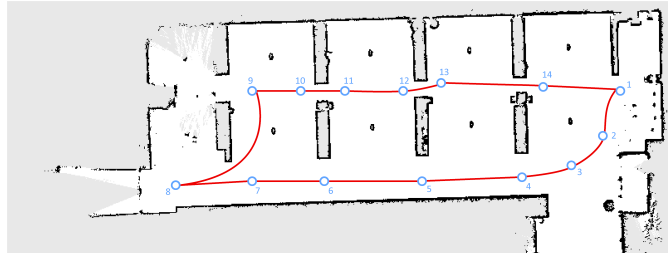


Figure 4.12: Visual representation of waypoints in the office.

the platform navigated flawlessly through the waypoints, completing three cycles in 38 minutes. Given the stop time at every waypoint (about 5 seconds) and the distance travelled, the duration of the execution is considered satisfactory, enabling the robot to complete at least three missions with a full battery (expected battery time is 2 hours from full capacity). By exploiting the keep out filter, it

was possible to avoid exploration in areas that are considered to be "dangerous" for a flawless mission execution: due to hardware limitations represented by the LiDAR's positioning on the robot, criticalities were found while travelling near office chairs, characterized by the five-star caster base and a spindle. The platform is able to recognize the spindle as an obstacle, but cannot account for the base, which farther extends the chair's footprint. To avoid collision with the base, the keep out overlay of the working stations was enlarged to account for the chairs, greatly improving the platform's navigation. Such difficulty could be overcome with the implementation of a second LiDAR scanner, positioned under the platform's base, to account for obstacles in tight proximity with the floor. This solution would also be preferred with respect to the implementation of a 3D LiDAR scanner, which would raise problems related to its installation on the platform and its cost.

Below, the result of the mission is presented in the form of a heatmap, complete of all three exploration cycles.

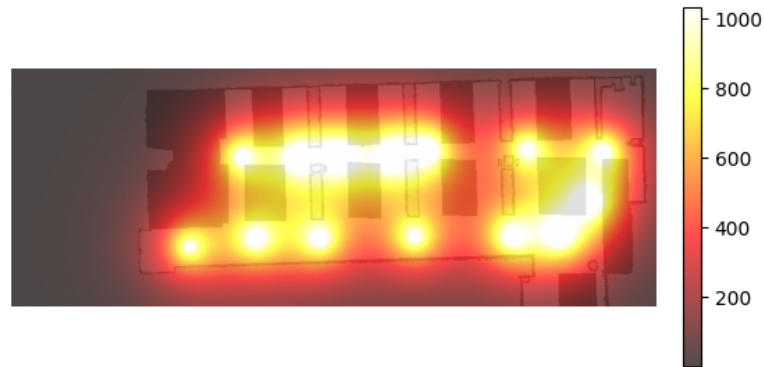


Figure 4.13: Heatmap of ISPIC's working space.

Chapter 5

Conclusion and Future Work

In this thesis, a sensor-informed navigation strategy was proposed to enable mobile sanitization in indoor workplaces at reduced cost compared to fully fixed sanitization infrastructures. Among the various pollutants and tracers that can be monitored in enclosed environments, CO_2 was selected as the primary indicator to guide exploration, based on its wide use in the literature as a biomarker for ventilation adequacy and human presence. Map-based navigation on a pre-acquired 2D occupancy grid was adopted as a robust solution for long-running missions, reflecting the typical operating conditions of a service robot repeatedly patrolling the same structured environment.

The mission was formalized as a patrolling routine in which the system sequentially visits a set of predefined waypoints and, at each location, collects CO_2 measurements to estimate the local Indoor Air Quality (IAQ). Once all waypoints in the mission have been explored, the system sorts them according to the measured CO_2 levels and selects the waypoint with the highest concentration as the starting point for the next patrol cycle, while preserving the continuity of the waypoint sequence. In this way, subsequent missions are biased toward areas that exhibit persistently elevated CO_2 which are likely associated with higher occupancy or poorer ventilation.

To realize this behavior, the system was implemented on top of the ROS 2 middleware, leveraging its communication primitives and the Nav2 stack for navigation. The mission logic was encoded using a Behavior Tree architecture, implemented through a C++ BT library and extended with ROS 2-specific wrappers that allow BT Action nodes to act as action clients via plugins. During execution, the BT interacts with a backend processing unit—the “behavior tree server”—via actions and services. This server is responsible for loading and managing waypoint lists,

sampling and averaging CO₂ at each waypoint, performing interpolation to construct a CO₂ heatmap, saving diagnostic data, and sorting waypoints according to the most recent measurements. Auxiliary nodes manage BT ticking and injection into the execution, as well as parallel conditions such as battery monitoring and button-driven mode switching, improving robustness and operator control.

The proposed architecture was integrated into a heavy-duty mobile platform designed to accommodate the mass and volume of a HEPA-based sanitization unit. The platform was dimensioned to sustain loads comparable to fixed sanitization devices, while preserving sufficient maneuverability for indoor environments. Experimental results in simulated and real office-like scenarios showed that the robot can autonomously complete sanitization missions, prioritize CO₂-dense areas of the patrol loop, and reveal a clear correlation between elevated CO₂ levels and human presence, the dominant source of CO₂ in indoor spaces.

The strength of the proposed architecture lies in its scalability and modularity. Any computational unit capable of running ROS 2 can host the same software stack, and any sensor, actuator, or human-machine interface that exposes data and commands via ROS topics, services, or actions can be integrated with limited effort. This opens the door to richer IAQ monitoring configurations in which multiple environmental sensors (e.g., particulate matter, VOCs, temperature, humidity) are combined to provide a more complete picture of indoor air quality. A natural extension would be to allow the user to select, via hardware buttons or a graphical interface, which sensor or subset of sensors should drive the analysis and visualization. In this scenario, multiple sensor-specific heatmaps could be generated and logged, enabling more advanced post-hoc analysis of how different pollutants respond to sanitization and ventilation strategies.

A further step would be to perform sensor fusion, combining heterogeneous sensor readings into a single IAQ index through an appropriate weighting scheme or learned model and interpolating this index over space. This would yield a unified heatmap encoding the joint behavior of all monitored quantities, at the cost of additional modeling complexity that lies beyond the scope of the present work. On the hardware side, the integrated platform could be improved by introducing a dedicated charging dock and a unified power architecture: a single shared battery powering the compute unit, actuators, and sanitization module would simplify on-site operation and avoid continuous battery swapping. At the current stage, only the actuators' battery level is monitored; extending energy monitoring and management to the entire system is an important step toward fully autonomous, continuous sanitization missions.

Appendix A

Probabilistic Localization Background

The following section largely owes credit to what it has been presented in [39]. A central problem in mobile robotics is enabling the robot to maintain an estimate of its pose within an environment. Given a known map, **localization** is the problem of estimating the robot’s pose x_t (typically position and orientation) in a global reference frame, using noisy control data u_t (proprioceptive) and measurement data z_t (exteroceptive).

In a probabilistic formulation, localization is treated as a **state estimation** problem. The robot’s internal knowledge about its pose is represented as a belief over the state space,

$$bel(x_t) = p(x_t | z_{1:t}, u_{1:t}),$$

which encodes the full posterior distribution over poses given all measurements and controls up to time t . This must be distinguished from the (unknown) **true state** x_t ; because sensors and actuators are noisy, the belief is never a point estimate but a distribution.

The standard tool to recursively compute this belief is the *Bayes filter*, which updates the posterior as new controls and measurements arrive. However, exact Bayes filtering is rarely tractable for realistic, continuous, high-dimensional state spaces such as robot poses in a map. This motivates approximate filtering algorithms, such as Gaussian filters and nonparametric filters. In this thesis, broader importance is given to nonparametric filter, and in particular **particle filters**, which form the basis of the *Monte Carlo Localization* (MCL) algorithm and its adaptive variants employed in *Nav2*.

Recursive State Estimation and the Bayes Filter

The *Bayes filter* represents the main approach when dealing with the computation of belief, as previously stated. The algorithm assumes a Markovian dynamical system with state x_t , control u_t and measurement z_t , a motion model $p(x_t|u_t, x_{t-1})$, and a measurement model $p(z_t|x_t)$. The filter is available in the table below.

Algorithm 2 The general algorithm for Bayes filtering.

```

1: Algorithm Bayes_filter( $bel(x_{t-1}), u_t, z_t$ ):
2: for all  $x_t$  do
3:    $bel(x_t) = \int p(x_t | u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1}$ 
4:    $bel(x_t) = \eta p(z_t | x_t) bel(x_t)$ 
5: end for
6: return  $bel(x_t)$ 
    
```

Under the *Markov assumption* (state as a complete summary of the past), belief is updated in two steps:

1. **Prediction:** using the the previous belief and the motion model,

$$\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1}$$

2. **Correction:** incorporating the new observation,

$$bel(x_t) = \eta p(z_t|x_t) \overline{bel}(x_t),$$

where η is a normalizing constant.

Clearly, a recursive computation of this sort requires a "starting point", therefore an initial belief $bel(x_0)$ must be specified as boundary condition. For localization, this is often:

- A point mass centered at a known initial pose (localization with known start),
or
- A uniform distribution over all feasible poses (global localization).

This formulation is conceptually clean and completely general. However, exact implementation is only feasible in discrete and low-dimensional state spaces (finite sums instead of integrals) and in linear-Gaussian models, where the Bayes filter reduces to a (Extended/Unscented) Kalman filter. For mobile robot localization on continuous SE(2) or SE(3) state spaces, with nonlinear motion and complex, often multimodal measurement likelihoods, these integrals cannot be solved analytically,

nor represented exactly. Hence, the *Bayes filter* is mainly a conceptual reference, given the required computational complexity to account for all the variables that may cause violations to the *complete state* (Markov) *assumption*. In practice, approximations with either parametric or nonparametric filters are employed.

A.1 Particle Filters: A Nonparametric Approximation

Particle filters provide a nonparametric approximation of the Bayes filter by representing the belief $bel(x_t)$ as a finite set of samples (particles):

$$\mathcal{X}_t = \{x_t^{[m]}\}_{m=1}^M,$$

optionally with associated weights $w_t^{[m]}$. Each particle is a hypothesis about the current state. Intuitively, regions of state space with high posterior probability are represented by a higher density of particles. A standard particle filter algorithm can be dissected in the following steps:

1. **Sampling/prediction:** for each previous particle $x_{t-1}^{[m]}$, draw a predicted state

$$x_t^{[m]} \sim p(x_t|u_t, x_{t-1}^{[m]})$$

2. **Importance weightinh:** compute the importance weight based on the measurement likelihood:

$$w_t^{[m]} \propto p(z_t|x_t^{[m]})$$

3. **Resampling:** draw M new particles from the weighted set, with probability proportional to $w_t^{[m]}$. The concept of this step can be translated as "survival of the fittest" and consists in concentrating particles in regions of high posterior probability and discarding unlikely hypotheses.

A standard particle filter algorithm is provided below.

Algorithm 3 The particle filter algorithm, a variant of the Bayes filter based on importance sampling.

```

1: Algorithm Particle_filter( $\mathcal{X}_{t-1}, u_t, z_t$ ):
2:  $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$ 
3: for  $m \leftarrow 1$  to  $M$  do
4:   sample  $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$ 
5:    $w_t^{[m]} \leftarrow p(z_t \mid x_t^{[m]})$ 
6:    $\bar{\mathcal{X}}_t \leftarrow \bar{\mathcal{X}}_t \cup \{x_t^{[m]}, w_t^{[m]}\}$ 
7: end for
8: for  $m \leftarrow 1$  to  $M$  do
9:   draw index  $i$  with probability  $\propto w_t^{[i]}$ 
10:  add  $x_t^{[i]}$  to  $\mathcal{X}_t$ 
11: end for
12: return  $\mathcal{X}_t$ 
    
```

Under suitable assumption and as $M \rightarrow \infty$, this procedure converges to the exact Bayes posterior. In practice, for finite M , the filter provides good approximation if enough particles are used. Particle filters inherit the conceptual elegance of the Bayes filter while avoiding its intractable integrals, but they introduce their own characteristic limitations. Understanding these limitations is essential to motivate *Monte Carlo Localization* and, ultimately, *Adaptive Monte Carlo Localization*.

From a theoretical standpoint, particle filters approximate the continuous belief $bel(x_t)$ with a finite set of random samples. As long as the number of particles is large, this approximation converges asymptotically to the true posterior. However, in a real robotic system the number of particles is always finite and constrained by computation, so the filter necessarily suffers from sampling variance. In practice, this means that two iterations of the same algorithm, with the same models and data, can produce slightly different beliefs as different random samples are drawn. When the number of particles is low with respect to the "spread" and dimensionality of the state space, these random fluctuations become visible as noisy, sometimes distorted approximations of the true posterior. Increasing the number of particles reduces this variance but immediately raises computational costs, especially in localization where evaluating the sensor model for each particle can be expensive.

The resampling step, which is essential for rejecting unlikely hypotheses and concentrating particles where the posterior is large, may act as a double-edged sword. Resampling repeatedly redistributes computational effort toward particles with high importance weights, which is desirable, but it gradually erodes diversity in the particle set. Over time, many particles become duplicates of a few highly

weighted ancestors, while other regions of the state space are no longer represented at all. In benign situations, this loss of diversity is harmless, as the belief is indeed concentrated around one pose; in more difficult situations — such as global localization, symmetric environments, or after large unmodeled disturbances — the lack of diversity becomes problematic: the filter may confidently track the wrong mode and be unable to "rediscover" the correct pose because no particles are left near it. This phenomenon is often referred to as particle depletion or particle deprivation.

A related issue is bias introduced by the finite particle set and the normalization implicit in resampling. In the extreme case of very few particles, the measurement information is not properly reflected in the distribution, and the filter may behave more like a noisy predictor driven mainly by the motion model. Even with more particles, if they are poorly distributed after resampling, the subsequent prediction and weighting steps are forced to work with a distorted representation of the posterior. The quality of the approximation is thus intimately tied to both the number of particles and the resampling strategy. Sophisticated resampling schemes (such as *low-variance resampling*) mitigate variance and preserve diversity better than naive schemes, but they do not remove the fundamental trade-off: using more particles improves accuracy and robustness, but at the cost of computation.

These limitations are particularly visible on mobile robot localization. To solve the global localization problem, the particle filter must initially represent a belief that is roughly uniform over all feasible poses. Covering such a large state space adequately requires a very large number of particles to avoid leaving vast portions of the map unexplored and misinterpreting certain hypotheses despite their consistency with sensor data. Once the robot has converged to a small region of the map, however, such a large particle set becomes wasteful. Most particles then occupy states that are very similar to each other, and the filter spends a significant amount of computation resampling almost identical hypotheses. If the number of particles is instead chosen small from the beginning to save time, the filter becomes fragile: it may fail to localize globally, or it may collapse to an incorrect pose and lack the diversity of need for recovery.

A.2 Monte Carlo Localization

Monte Carlo Localization (MCL) specializes particle filtering to the mobile robot localization problem in a known map m . The state is the robot pose,

$$x_t = (x, y, \theta),$$

and the belief $bel(x_t)$ is represented by a set of pose particles. In this application, the motion model $p(x_t|u_t, x_{t-1})$ describes how controls or odometry u_t move the robot

from x_{t-1} to x_t , including noise (odometry motion model), while the measurement model $p(z_t|x_t, m)$ represents the probability of observing sensor data z_t (e.g. laser scan) from pose x_t given the map m . The last model strongly constrains pose hypotheses by comparing expected and actual sensor readings. The algorithm supports global localization by initializing particle uniformly over the feasible pose space, along with position tracking by concentrating particles around a known starting pose. An important advantage is the ability to represent multi-modal belief (e.g. symmetric environments, kidnapped robot situations) that are difficult for unimodal Gaussian filters. All the previously discussed limitations carry over: global localization pushes toward many particles to obtain adequate coverage of the map, fine-grained scan matching pushes toward complex, potentially expensive sensor models, and long-term operation in dynamic environments stresses the filter's ability to maintain diversity and recover from occasional catastrophic errors. In a fixed-size MCL implementation, the user must select a single particle count that balances these competing needs, which is rarely optimal across all operating phases.

A.3 Adaptive Monte Carlo Localization

To alleviate the trade-off between robustness and efficiency in fixed-sized MCL, *Adaptive Monte Carlo Localization* (AMCL) varies the number of particles over time. The central intuition is that a spread-out, multi-modal belief — typical at the beginning of global localization or after a kidnapping — requires many particles to represent all plausible hypotheses. Conversely, once the robot is well localized and the posterior is sharply concentrated, only a relatively small number of particles is needed to maintain an accurate approximation. Instead of choosing a global, static particle count, AMCL adjusts the number of particles online, striving to maintain a prescribed approximation quality while avoiding unnecessary computation.

KLD-sampling gives a principled way to formalize this adaptation. Rather than guessing what "enough particles" means, the method specifies a desired bound on the approximation error between the sample-based belief and the (unknown) true posterior, measured via the Kullback-Leibler divergence. The user sets two parameters: an accuracy ϵ , controlling the tolerable divergence, and a confidence level $1 - \delta$, expressing the probability that the approximation lies within this bound. During resampling, the algorithm effectively counts how many distinct regions of the state space are being represented by the particles. Intuitively, the more bins are occupied, the more complex the belief, and the more samples are needed to represent it with the desired accuracy. Using results from statistical learning theory, KLD-sampling computes the minimal number of particles required for the current number of occupied bins to satisfy the given (ϵ, δ) constraint. Sampling continues until this threshold is reached, after which resampling stops and the algorithm

proceeds. The algorithm is present below.

Algorithm 4 KLD_Sampling_MCL($\mathcal{X}_{t-1}, u_t, z_t, m, \epsilon, \delta$)

```

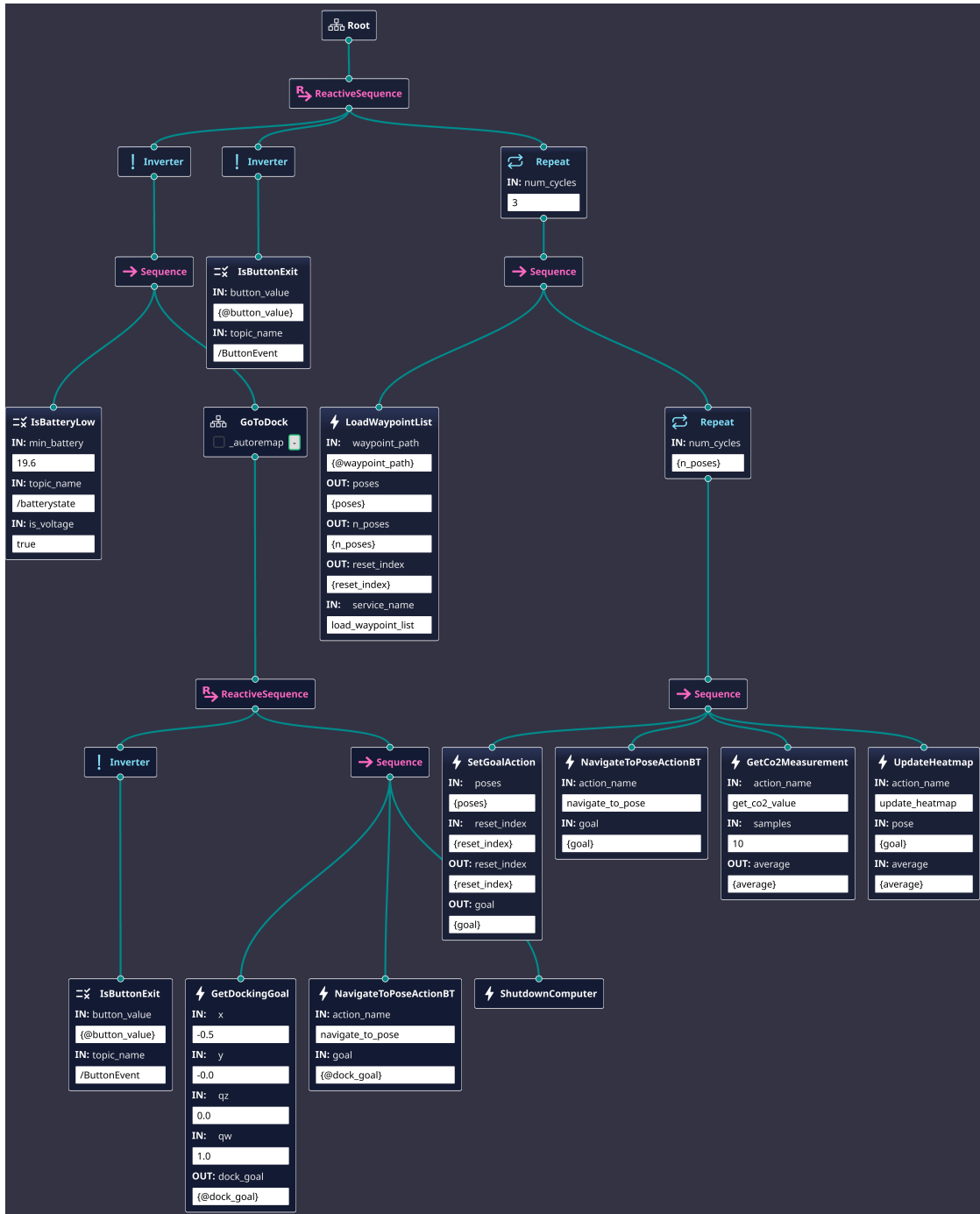
1:  $\mathcal{X}_t = \emptyset$ 
2:  $M = 0, M_x = 0, k = 0$ 
3: for all  $b \in H$  do
4:    $b = \text{empty}$ 
5: end for
6: while  $M \leq M_{\mathcal{X}} M \leq M_{\mathcal{X}_{\min}}$  do
7:   draw  $i$  with probability  $\propto w_{t-1}^{[i]}$ 
8:    $x_t^{[M]} = \text{SAMPLE\_MOTION\_MODEL}(u_t, x_{t-1}^{[i]})$ 
9:    $w_t^{[M]} = \text{MEASUREMENT\_MODEL}(z_t, x_t^{[M]}, m)$ 
10:   $\mathcal{X}_t = \mathcal{X}_t + \langle x_t^{[M]}, w_t^{[M]} \rangle$ 
11:  if  $x_t^{[M]}$  falls into empty bin  $b$  then
12:     $k = k + 1$ 
13:     $b = \text{non-empty}$ 
14:    if  $k > 1$  then
15:       $M_{\mathcal{X}} \leftarrow \frac{k-1}{2\epsilon} \left( 1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}} z_{1-\delta} \right)^3$ 
16:    end if
17:  end if
18:   $M = M + 1$ 
19: end while
20: return  $\mathcal{X}_t$ 

```

Adaptive MCL as used in ROS Nav2 builds on this idea (together with additional heuristics, such as injecting random particles for recovery), enabling efficient global localization and tracking while keeping computational cost under control.

Appendix B

WaypointsBattery - Complete Tree



Bibliography

- [1] *Robotics — Vocabulary*. 2021. URL: <https://www.iso.org/obp/ui/#iso:std:iso:8373:ed-3:v1:en> (visited on 02/03/2026) (cit. on p. 1).
- [2] In Lee. «Service robots: a systematic literature review». In: *Electronics* 10.21 (2021), p. 2658 (cit. on pp. 1, 2).
- [3] Juan Angel Gonzalez-Aguirre, Ricardo Osorio-Oliveros, Karen L Rodríguez-Hernández, Javier Lizárraga-Iturralde, Ruben Morales Menendez, Ricardo A Ramirez-Mendoza, Mauricio Adolfo Ramirez-Moreno, and Jorge de Jesus Lozoya-Santos. «Service robots: Trends and technology». In: *Applied Sciences* 11.22 (2021), p. 10702 (cit. on pp. 1, 2).
- [4] *Service Robotics Market Size, By Component, By Type, By Mobility, By Technology, By Application, Forecast 2025 - 2034*. 2025. URL: <https://www.gminsights.com/industry-analysis/service-robotics-market-size> (visited on 02/03/2026) (cit. on p. 2).
- [5] Steve Macenski, Tom Moore, David V Lu, Alexey Merzlyakov, and Michael Ferguson. «From the desks of ROS maintainers: A survey of modern & capable mobile robotics algorithms in the robot operating system 2». In: *Robotics and Autonomous Systems* 168 (2023), p. 104493 (cit. on p. 2).
- [6] World Health Organization et al. «Combined or multiple exposure to health stressors in indoor built environments: an evidence-based review prepared for the WHO training workshop “Multiple environmental exposures and risks”: 16–18 October 2013, Bonn, Germany». In: (2014) (cit. on p. 2).
- [7] Alessandra Cincinelli and Tania Martellini. *Indoor air quality and health*. 2017 (cit. on pp. 3, 10).
- [8] Usha Satish, Mark J Mendell, Krishnamurthy Shekhar, Toshifumi Hotchi, Douglas Sullivan, Siegfried Streufert, and William J Fisk. «Is CO₂ an indoor pollutant? Direct effects of low-to-moderate CO₂ concentrations on human decision-making performance». In: *Environmental health perspectives* 120.12 (2012), p. 1671 (cit. on p. 3).

- [9] Deepanjan Majumdar. «Modelling accumulation of respiratory-CO₂ in closed rooms leading to decision-making on room occupancy». In: *MAPAN-Journal of metrology society of India* (2020) (cit. on p. 3).
- [10] William J Fisk and Arthur H Rosenfeld. «Estimates of improved productivity and health from better indoor environments». In: *Indoor air* 7.3 (1997), pp. 158–172 (cit. on p. 3).
- [11] Donald K Milton, P Mark Glencross, and Michael D Walters. «Risk of sick leave associated with outdoor air supply rate, humidification, and occupant complaints». In: (2000) (cit. on p. 3).
- [12] SN Rudnick and Donald K Milton. «Risk of indoor airborne infection transmission estimated from carbon dioxide concentration.» In: *Indoor air* 13.3 (2003) (cit. on p. 3).
- [13] Zhe Peng and Jose L Jimenez. «Exhaled CO₂ as a COVID-19 infection risk proxy for different indoor environments and activities». In: *Environmental Science & Technology Letters* 8.5 (2021), pp. 392–397 (cit. on p. 3).
- [14] Dadi Zhang and Philomena M Bluysen. «Exploring the possibility of using CO₂ as a proxy for exhaled particles to predict the risk of indoor exposure to pathogens». In: *Indoor and Built Environment* 32.10 (2023), pp. 1958–1972 (cit. on p. 3).
- [15] Ishaan Mehta, Hao-Ya Hsueh, Sharareh Taghipour, Wenbin Li, and Sajad Saeedi. «UV disinfection robots: a review». In: *Robotics and autonomous systems* 161 (2023), p. 104332 (cit. on pp. 5, 6, 16).
- [16] Alyssa Pierson, John W Romanishin, Hunter Hansen, Leonardo Zamora Yanez, and Daniela Rus. «Designing and deploying a mobile UVC disinfection robot. In 2021 IEEE». In: *RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6700–6707 (cit. on pp. 5, 6).
- [17] Dragoş-Vasile Bratu, Maria-Alexandra Zolya, and Sorin-Aurel Moraru. «Robo-CoV Cleaner: An indoor autonomous UV-C disinfection robot with advanced dual-safety systems». In: *Sensors* 24.3 (2024) (cit. on p. 6).
- [18] Zifan Yao, Na Ma, and Youdong Chen. «An autonomous mobile combination disinfection system». In: *Sensors* 24.1 (2023), p. 53 (cit. on pp. 6, 7).
- [19] Joachim Curtius, Manuel Granzin, and Jann Schrod. «Testing mobile air purifiers in a school classroom: Reducing the airborne transmission risk for SARS-CoV-2». In: *Aerosol Science and Technology* 55.5 (2021), pp. 586–599 (cit. on p. 7).
- [20] Haoguang Yang et al. «Occupant-centric robotic air filtration and planning for classrooms for Safer school reopening amid respiratory pandemics». In: *Robotics and Autonomous Systems* 147 (2022), p. 103919 (cit. on p. 7).

- [21] Finn Felix Duill, Florian Schulz, Aman Jain, Berend van Wachem, and Frank Beyrau. «Comparison of portable and large mobile air cleaners for use in classrooms and the effect of increasing filter loading on particle number concentration reduction efficiency». In: *Atmosphere* 14.9 (2023), p. 1437 (cit. on p. 7).
- [22] Mohet Narang, Mehul Rana, Jai Patel, Steve D'souza, Princewill Onyechie, Cody Berry, Mohsen Tayefeh, and Ahmad Barari. «Fighting covid: an autonomous indoor cleaning robot (aicer) supported by artificial intelligence and vision for dynamic air disinfection». In: *2021 14th IEEE international conference on industry applications (INDUSCON)*. IEEE. 2021, pp. 1146–1153 (cit. on pp. 8, 16).
- [23] Pawel Idziak and Michal Gojtowski. «Smart air purifier suitable for small public spaces». In: *ITM web of conferences*. Vol. 28. EDP Sciences. 2019, p. 01015 (cit. on p. 9).
- [24] M Anitha and Lakshmi Sutha Kumar. «Development of an IoT-enabled air pollution monitoring and air purifier system». In: *MAPAN* (2023), p. 1 (cit. on p. 9).
- [25] M. Rudra Kumar, Tigulla Hruthika Goud, Bodula Sujitha, and Kanithi Ravi Kishore. «AI Powered Cleaning Robot». In: *International Journal For Multidisciplinary Research* 6.3 (2024) (cit. on p. 9).
- [26] Francesco Salamone, Lorenzo Belussi, Ludovico Danza, Theodore Galanos, Matteo Ghellere, and Italo Meroni. «Design and development of a nearable wireless system to control indoor air quality and indoor lighting quality». In: *Sensors* 17.5 (2017), p. 1021 (cit. on pp. 10, 15).
- [27] Gonalo Marques, Ivan Miguel Pires, Nuno Miranda, and Rui Pitarma. «Air quality monitoring using assistive robots for ambient assisted living and enhanced living environments through internet of things». In: *Electronics* 8.12 (2019), p. 1375 (cit. on pp. 10, 13, 15).
- [28] Ming Jin, Shichao Liu, Stefano Schiavon, and Costas Spanos. «Automated mobile sensing: Towards high-granularity agile indoor environmental quality monitoring». In: *Building and Environment* 127 (2018), pp. 268–276 (cit. on pp. 10–12, 14, 15).
- [29] Shuichi Tamura, Ouki Kato, and Susumu Shirayama. «Development of indoor CO2 monitoring system with an integrated fixed sensor and a mobile measuring robot». In: *ASME Journal of Engineering for Sustainable Buildings and Cities* 6.1 (2025), p. 011001 (cit. on pp. 10, 13–16).
- [30] Prajwala Srivatsa and Amit Pandhare. «Indoor air quality: IoT solution». In: *Proceedings of the National Conference NCPCL*. Vol. 2016. 2016, p. 19 (cit. on p. 10).

- [31] Alessia Di Gilio, Jolanda Palmisani, Manuela Pulimeno, Fabio Cerino, Mirko Cacace, Alessandro Miani, and Gianluigi de Gennaro. «CO2 concentration monitoring inside educational buildings as a strategic tool to reduce the risk of Sars-CoV-2 airborne transmission». In: *Environmental research* 202 (2021), p. 111560 (cit. on pp. 10, 15).
- [32] Jordi Palacín, Elena Rubies, and Eduard Clotet. «A retrospective analysis of indoor CO2 measurements obtained with a mobile robot during the COVID-19 pandemic». In: *Sensors* 24.10 (2024), p. 3102 (cit. on pp. 10, 12, 14–16).
- [33] Edgars Edelmers et al. «Pilot Study on the Physical, Chemical, and Biological Determinants of Indoor Air Quality in University Classrooms». In: *Rīgas Tehniskās Universitātes Zinātniskie Raksti* 27.1 (2023), pp. 753–762 (cit. on p. 10).
- [34] Yuan Yang, Jiang Liu, Wei Wang, Yu Cao, and Heng Li. «Incorporating SLAM and mobile sensing for indoor CO2 monitoring and source position estimation». In: *Journal of Cleaner Production* 291 (2021), p. 125780 (cit. on pp. 11, 15, 16).
- [35] Anggiat Mora Simamora, Asep Denih, and Mohamad Iqbal Suriansyah. «Indoor Air Quality Detection Robot Model Based on the Internet of Things (IoT)». In: *arXiv preprint arXiv:2505.19600* (2025) (cit. on pp. 11, 15).
- [36] L Abdallah and A Nasr. «Using robots to improve indoor air quality and reduce COVID-19 exposure». In: *Journal of applied research and technology* 19.3 (2021), pp. 227–237 (cit. on p. 12).
- [37] Patrizia Ribino, Marina Bonomolo, Carmelo Lodato, and Gianpaolo Vitale. «A humanoid social robot based approach for indoor environment quality monitoring and well-being improvement». In: *International Journal of Social Robotics* 13.2 (2021), pp. 277–296 (cit. on p. 13).
- [38] Nav2. *Nav2 Official Website*. URL: <https://docs.nav2.org> (visited on 02/27/2026) (cit. on pp. 17, 20).
- [39] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics / Sebastian Thrun, Wolfram Burgard, Dieter Fox*. eng. Intelligent robotics and autonomous agents. Cambridge, Massachusetts ; MIT Press, 2006 - 2006. ISBN: 0262201623 (cit. on pp. 17, 59).
- [40] Michele Colledanchise and Petter Ögren. *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018 (cit. on p. 18).
- [41] BehaviorTree. *BehaviorTree.ROS2*. URL: <https://github.com/BehaviorTree/BehaviorTree.ROS2> (visited on 02/27/2026) (cit. on p. 20).
- [42] *BehaviorTree.CPP website*. URL: <https://www.behaviortree.dev/> (visited on 02/26/2026) (cit. on p. 20).

- [43] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. «Robot Operating System 2: Design, architecture, and uses in the wild». In: *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074. URL: <https://www.science.org/doi/abs/10.1126/scirobotics.abm6074> (cit. on pp. 21, 32).
- [44] Shanghai Slamtec Co., Ltd. *RPLIDAR A1 Low Cost 360 Degree Laser Range Scanner (Model A1M8)*. Tech. rep. Datasheet rev. 3.0. Shanghai, China: Shanghai Slamtec Co., Ltd., Oct. 2020. URL: https://bucket-download.slamtec.com/d1e428e7efbdcd65a8ea111061794fb8d4ccd3a0/LD108_SLAMTEC_rplidar_datasheet_A1M8_v3.0_en.pdf (cit. on p. 25).
- [45] Infineon Technologies. *XENSIV™ PAS CO2 1.5*. Datasheet V1.1. Infineon Technologies, May 2024. URL: https://www.mouser.com/catalog/specsheets/Infineon_5-27-2024_DS_XENSIVTM_PASC02_V15.pdf (cit. on p. 26).
- [46] Jongwon Kwon, Gwanghoon Ahn, Gysik Kim, Jo Chun Kim, and Hiesik Kim. «A study on NDIR-based CO2 sensor to apply remote air quality monitoring system». In: *2009 ICCAS-SICE*. 2009, pp. 1683–1687 (cit. on p. 27).
- [47] *RoboMaster M3508 P19 Brushless DC Gear Motor V1.0*. User Guide. SZ DJI Technology Co., Ltd. Aug. 2017. URL: <https://rm-static.djicdn.com/tem/17348/RoboMaster%20M3508%20P19%20Brushless%20DC%20Gear%20Motor%20V1.0.pdf> (cit. on p. 28).
- [48] *RoboMaster C620 Brushless DC Motor Speed Controller V1.01*. User Guide. SZ DJI Technology Co., Ltd. Sept. 2017. URL: <https://rm-static.djicdn.com/tem/17348/RoboMaster%20C620%20Brushless%20DC%20Motor%20Speed%20Controller%20V1.01.pdf> (cit. on p. 28).
- [49] *ASPRA Lite S400: Product Details (Datasheet)*. English datasheet; order code VFA-010103. VFA Solutions. Schiedam, NL. URL: <https://www.vfa-solutions.com/wp-content/uploads/2021/08/VFA-Solutions-Datasheet-ASPRA-Lite-S400-EN.pdf> (cit. on p. 31).