Hugo André
Costa Barros

Navegação para Desinfeção de Espaços Públicos

Navigation for Disinfection of Public Spaces

Universidade de Aveiro
2023

Hugo André Costa Barros

Navegação para Desinfeção de Espaços Públicos

Navigation for Disinfection of Public Spaces

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica do Doutor José Nuno Panelas Nunes Lau, Professor Associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Eurico Farinha Pedrosa, Investigador Doutorado do Instituto de Engenharia Eletrónica e Informática de Aveiro - IEETA da Universidade de Aveiro.

Esta dissertação contou com o apoio do projeto "Automated Germicidal Irradiation System" com a referência POCI-01-0247-FEDER-072237 e foi financiado pelo Fundo Europeu de Desenvolvimento Regional (FEDER).

o júri / the jury

presidente / president

Prof. Doutor Rui Manuel Escadas Ramos Martins
Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade do Aveiro

vogais / examiners committee

Prof. Doutor Luís Paulo Gonçalves dos Reis
Professor Associado da Faculdade de Engenharia da Universidade do Porto

Prof. Doutor José Nuno Panelas Nunes Lau
Professor Associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade do Aveiro

agradecimentos / acknowledgements

Quero começar por agradecer aos meus pais, Manuela e Paulo, por acreditarem sempre em mim e me apoirem nos momentos mais dificies. Agradeço ao meu irmão David e avó Sãozinha pelo carinho e apoio.

Um agradecimento especial ao Professor Doutor José Nuno Panelas Nunes Lau e ao Doutor Eurico Farinha Pedrosa pela paciência e por toda a ajuda prestada durante esta dissertação.

A todos os meus amigos que me acompanharam ao longo destes anos. Juntos partilhamos momentos inesqueciveis que irei levar para a vida.

Por fim, gostaria de agradecer à minha companheira Patrícia, sem ela nada disto teria sido possivel. Obrigado por toda a paciência, apoio e motivação que me deste.

Palavras Chave

ROS, Planeamento de Caminhos de Cobertura, Navegação de Robôs, Localização em Espaços Interiores, MiR100, Desinfeção Móvel, Sensor Ultravioleta.

Resumo

Este documento descreve um projeto destinado a melhorar as soluções actuais para o planeamento do percurso em cenários de desinfeção. O projeto propõe o desenvolvimento de um sistema automatizado de irradiação germicida utilizando um robô móvel para a desinfeção de espaços públicos. A dissertação centra-se na investigação e desenvolvimento de um sistema de navegação autónomo integrado no robô, analisando algoritmos de localização e navegação. Pretende-se desenvolver um sistema capaz de desinfetar espaços interiores utilizando o robô e um sensor ultravioleta, tendo como objectivos o estudo dos métodos de navegação existentes, o desenvolvimento de um metodo para visualizar a area irradiada pelo sensor UV, e o teste e validação das técnicas desenvolvidas.

É discutido o desenvolvimento de um sistema de desinfeção de espaços públicos com base no uso da framework Robot Operating System (ROS). Para a construção do sistema, não só foram utilizados pacotes ROS disponiveis online assim como foram criados pacotes ROS especicos do projeto. Destaca-se a integração com o ROS do robo MiR100, o método de localização baseado em scan-matching, o planeamento da navegação usando como planeador global o algoritmo backtracking spiral (BSA) e como planeador local um controlador PID, e a representação da área de cobertura do Sensor ultravioleta. O documento também menciona os parâmetros e componentes envolvidos na operação do sistema e a importância da cobertura abrangente para fins de desinfeção.

O passo seguinte foi testar o sistema desenvolvido, o cenário de teste e os critérios de seleção utilizados para avaliar a eficiência da desinfeção e afinar o sistema são abordados. As métricas utilizadas para avaliar o comportamento da desinfeção incluem a área desinfectada, o tempo de desinfeção, a sobreposição da área irradiada e o tempo de exposição à irradiação. São avaliados os parâmetros que influenciam a eficácia e a eficiência de um robô de desinfeção e são apresentados os resultados de testes realizados num ambiente simulado. São identificadas as combinações ideais de parâmetros para obter a melhor cobertura e tempo de desinfeção. O documento conclui com a escolha final dos parâmetros com base nas metricas acima referidas sendo a sobreposição da área irradiada a metrica mais relevante.

Em conclusão, utilizando o MiR100, que é um robô móvel autónomo compacto e altamente manobrável capaz de transportar cargas até 100kg, equipado com um sensor UV, foi alcançada uma cobertura de desinfeção de 97,10% após 22 minutos e 19 segundos, com algumas áreas a serem irradiadas mais do que uma vez. A cobertura do mapeamento foi bem sucedida, exceto nas áreas próximas das paredes, que não foram tratadas devido a limitações no cálculo do percurso do algoritmo BSA.

Keywords ROS, Coverage Path Planning, Robot Navigation, Indoor Localization, MiR100, Mobile Disinfection, Ultraviolet Sensor.

Abstract

This document describes a project aimed at improving current solutions for optimal path planning in disinfection scenarios. The project proposes the development of an automated germicidal irradiation system using a mobile robot for disinfection of public spaces. The dissertation focuses on the research and development of an autonomous navigation system integrated into the robot, analyzing algorithms for localization and navigation. The goal is to develop a system that can disinfect indoor spaces using the robot and an ultraviolet sensor, with objectives including studying existing navigation methods, developing a UV sensor component, and testing and validating the techniques developed.

The development of a system for disinfecting public spaces based on the use of the Robot Operating System (ROS) framework is presented. To build the system, not only were ROS packages available online used, but project-specific ROS packages were also created. Highlights include the integration with the ROS of the MiR100 robot, the localization method based on scan-matching, navigation planning using the backtracking spiral algorithm (BSA) as the global planner and a PID controller as the local planner, and the representation of the ultraviolet sensor coverage area. The document also mentions the parameters and components involved in the system's operation and the importance of comprehensive coverage for disinfection purposes.

The next step was to test the developed system, the test scenario and selection criteria used to evaluate the efficiency of disinfection and tuning the system are aborded. The metrics used to evaluate the behavior of disinfection include the disinfected area, disinfection time, overlapping irradiated area, and irradiation exposure time. It is evaluated the parameters that impact the effectiveness and efficiency of a disinfection robot and presents the results of tests conducted in a simulated environment. The optimal combinations of parameters for achieving the best disinfection coverage and time are identified. The document concludes with the final choice of parameters based on the overlapping of irradiated area.

In conclusion, using the MiR100 which is a compact, highly maneuverable autonomous mobile robot capable of carring payloads up to 100kg equiped with a UV Sensor, a disinfection coverage of 97.10% was achieved after 22 minutes and 19 seconds, with some areas being irradiated more than once. The mapping coverage was successful, except for areas close to the walls, which were left untreated due to limitations in the BSA algorithm's path computation.

Contents

Contents i

List of Figures v

List of Tables vii

Glossary ix

1 Introduction 1

1.1 Motivation . . . 1

1.2 Objectives . . . 2

1.3 Thesis Outline . . . 2

2 Autonomous Navigation 3

2.1 Localization . . . 4

2.1.1 Adaptive Monte Carlo Localization . . . 5

2.1.2 Scan Matching-based Localization . . . 7

2.2 Mapping . . . 7

2.2.1 Occupancy Grid . . . 8

2.3 Global Path Planner . . . 8

2.3.1 A-Star Algorithm . . . 9

2.3.2 Search-based Planning Lattice . . . 10

2.4 Mapping Coverage . . . 13

2.4.1 Spiral Spanning Tree Coverage . . . 13

2.4.2 Backtracking Spiral Algorithm . . . 15

2.5 Local Path Planner . . . 18

2.5.1 Dynamic Window Approach . . . 18

2.5.2 Time Elastic Band . . . 19

2.6 The Localization and Mapping package . . . 20

3 Robot Operating System 23

3.1 Development Tools . . . 25
3.1.1 RViz . . . 25
3.1.2 rqt_graph . . . 26
3.2 Gazebo . . . 26

4 Robot Disinfection State of Art **29**
4.1 Real Robots Applications . . . 30
4.1.1 UltraBot . . . 30
4.1.2 Smart Cleaner . . . 31
4.1.3 Intelligent Disinfection Robot . . . 31
4.1.4 UVD Robot . . . 32
4.1.5 Ava's UV Disinfection Robot . . . 33
4.1.6 Comparison between Disinfection Robots . . . 33
4.2 Coverage Path Planning . . . 34
4.2.1 `explore_lite` package . . . 34
4.2.2 `full_coverage_path_planner` package . . . 35

5 Disinfection of Spaces **37**
5.1 Introduction . . . 37
5.2 MiR100 Robot's Overview . . . 37
5.2.1 The `mir_robot` repository . . . 38
5.3 Overall Architecture of the Project . . . 38
5.4 System Architecture . . . 39
5.4.1 Sensorization . . . 40
5.4.2 Localization . . . 43
5.4.3 Navigation . . . 43
5.4.4 Disinfectioned Area Coverage . . . 47
5.5 Summary . . . 48

6 Results and Troubleshooting **49**
6.1 Methodology of the tests . . . 49
6.1.1 Method of Evaluation of the Disinfection . . . 50
6.2 Evaluation of the UV Sensor's orientation and `tool_radius` parameter . . . 53
6.3 Evaluation of the MiR100's velocity during forward movement and directional changes 54
6.4 Evaluation of the UV Sensor's Radiation Range . . . 56
6.5 Evaluation of Disinfection . . . 57

7 Conclusion and future work **59**
7.1 Conclusions . . . 59

7.2 Future work 59

Bibliography **61**

List of Figures

2.1 Mobile Robot Navigation Blocks 3
2.2 Breadth-first search planner (left) and Example of task performed by Dijkstra's algorithm (right) [39]. 9
2.3 Path from the orange point to the purple point generated by A* algorithm [39]. 9
2.4 A 3D (x, y, θ) lattice with at most five forward actions for each state and no backward actions. A full set of actions is shown for states s_1. For every state, this set of actions is translated and rotated appropriately, and all actions intersecting obstacles are removed [44]. 11
2.5 Example of a High-resolution action spaces (Left) and a Low-resolution action spaces (Right) [44]. 12
2.6 Types of subdivision of a 2D-size cell into a D-size subcell. (a) A double-sided edge; (b) a single-sided edge; (c) node doubling at a disconnected cell. [48] 14
2.7 (a) Path deformation along a single-sided edge; (b) Crossing of spanning-tree edges. [48] 14
2.8 Full Spiral-STC Coverage. The figure possesses two unrealistic features, which were added for clarity. The tool size D is shown unrealistically large with respect to the work area size, and the covering tool path is shown curved while it is rectilinear according to the algorithm. [48] 15
2.9 Example of a virtual pipe [51]. 17
2.10 Large scenario with consideration of way-points and obstacles. [54] 20

3.1 ROS as a Meta-Operating System [59]. 23
3.2 Example of ROS Computation Graph Level 25
3.3 RViz 26
3.4 rqt_graph 26
3.5 Gazebo 27

4.1 UltraBot Robot [66] 30
4.2 Smart Cleaner Robot [68] 31
4.3 Intelligent Disinfection Robot [69] 32
4.4 UVD Robot [73] 33
4.5 Ava's UV Disinfection Robot [74] 33

5.1 Illustration of the Overall Architecture of the Project 38
5.2 System Architecture Diagram at ROS Computation Level 40
5.3 Field of view of the LiDAR sensors on the real MiR100 [78] 40
5.4 Field of view of the LiDAR sensors on the simulated MiR100 (left); Field of view of the front left sensor (middle); Field of view of the rear right sensor (right). 41
5.5 Field of view of the LiDAR sensors on the simulated MiR100 41
5.6 Example of a orientation of UV Sensor of 180º (left image), and a orientation of 90º (right image). 42
5.7 Position of a payload of 50kg on the MiR100 Robot [78] 43
5.8 Representation of the minimum traversable gap with and `robot_radius` parameter values. 44
5.9 Illustration of possible configuration for `robot_radius` and `tool_radius` parameters [79]. 45
5.10 Example of a coverage path generated using a `tool_radius` of 0.53115m (left image) and a `tool_radius` of 0.29m (right image). 45
5.11 Representation of the "carrot" strategy of `tracking_pid` [80]. 46
5.12 Representation of the projected global point (PGP) strategy of `tracking_pid` [80]. . . . 46

6.1 Test scenario used. 49
6.2 Testing Architecture Diagram at ROS Communication Level. 50
6.3 Example of situations where overlapping irradiated area can vary because of different `tool_radius` (Top Left=0.4106m; Top Right=0.53115m) or different `rangeUV` (Bottom Left=0.5m; Bottom Right=2m) 52
6.4 Environment state after disinfection. 58

List of Tables

4.1 Comparison between robots and their used algorithms. (Some of them keep their software confidential.) 34

6.1 Value range of the parameters manipulated during tests. 50
6.2 Percentage of disinfected area and the corresponding disinfection time for different values of `tool_radius` and `dirAngle`. 53
6.3 Results of tests for different values of `target_x_vel` and `target_yaw_vel` and candidate `tool_radius`/`dirAngle` combination of 0.29m/180º. 54
6.4 Results of tests for different values of `target_x_vel` and `target_yaw_vel` and candidate `tool_radius`/`dirAngle` combination of 0.53115m/0º. 55
6.5 Results of tests for different values of `target_x_vel` and `target_yaw_vel` and candidate `tool_radius`/`dirAngle` combination of 0.4106m/180º. 55
6.6 Results of tests for different values of `rangeUV` using the configuration of Set 1. 56
6.7 Results of tests for different values of `rangeUV` using the configuration of Set 2. 56
6.8 Comparison of the two final candidates' configurations advantages. 57

Glossary

GermIrrad	Automated Germicidal Irradiation System
FEUP	Faculty of Engineering of the University of Porto
LEPABE	Laboratory for Process Engineering, Environment, Biotechnology and Energy
UA	University of Aveiro
IEETA	Institute of Electronics and Informatics Engineering of Aveiro
KF	Kalman Filter
MCL	Monte Carlo Localization
IRIS Lab	Laboratory Intelligent Robotics and Intelligent Systems
GNSS	Global Navigation Satellite System
AMCL	Adaptive Monte Carlo Locatization
KLD	Kullback-Leibler distance
ICP	Iteractive Corresponding Point
EM	expectation maximization
GVG	generalized Voronoi graphs
FIFO	first-in-first-out
A*	A-Star Algorithm
ARA*	Anytime Reparing A-Star Algorithm
ADA*	Anytime Dynamic A-Star Algorithm
SBPL	Search-based Planning Lattice
Spiral-STC	Spiral Spanning Tree Coverage
DWA	Dynamic Window Approach
TEB	Timed Elastic Band
EB	Elastic Band
ROS	Robot Operating System
CPP	Coverage Path Planning
FCPP	Full Coverage Path Planner
BSA	Backtracking Spiral Algorithm
BPs	Backtracking Points
RLS	Reference Lateral Side
OLS	Opposite Lateral Side
MBF	move_base_flex
PID	Proportional-Integral-Derivative
SLAM	Simultaneous Localization and Mapping
MiR	Mobile Industrial Robots
UV	Ultraviolet

CHAPTER 1

Introduction

1.1 Motivation

A highly urbanized and interconnected world presents numerous challenges in various fields. One is public health, because the possibility of air-mediated microbial diseases becoming an epidemic is significantly high. Influenza and tuberculosis have been and continue to be major public health challenges, but recently, the severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2) aggravated the situation. Regular disinfection of public spaces is essential to prevent contamination, which can be performed by exposing the materials to radiation (e.g. ultraviolet) or through chemical agents. In order to make the disinfection process automatic, mobile robots can be used to transport and control disinfectant equipment. Considering this need, we propose this dissertation as an investigation project to analyze and test options for improving current solutions for optimal path planning in disinfection scenarios.

The research project "Automated Germicidal Irradiation System (GermIrrad)", promoted by the consortium formed by the company Spinner Dynamics, the Faculty of Engineering of the University of Porto (FEUP) through the Laboratory for Process Engineering, Environment, Biotechnology and Energy (LEPABE), and the University of Aveiro (UA) through the Institute of Electronics and Informatics Engineering of Aveiro (IEETA), has the primary objective of developing a system based on a mobile robot for disinfection of public spaces without the need of interrupting its regular activity. FEUP is in charge of the research of the disinfection method, while UA has the autonomous mobile navigation part of the project. The work described in this document was created as part of the GermIrrad project as a Master's thesis at the University of Aveiro in Electronic and Telecommunications Engineering.

This dissertation describes the research and development in a simulated environment of an autonomous navigation system that will be integrated into the robot MiR100 being developed as part of the GermIrrad project. It analyses the most common algorithms for autonomous navigation, focusing on two critical problems: localization and navigation for disinfection. These steps are necessary for the robot to act autonomously in the environment in which it will operate.

1.2 Objectives

The aim of this dissertation is to tackle the issue of regular disinfection of public spaces. The goal is to develop a system that is able to disinfect an indoor space using the robot MiR100 alongside a Ultraviolet (UV) sensor as method of disinfection. The principal objectives of this work are the following:

- Study of existing methods used in mobile navigation;
- Familiarization with navigation techniques for mapping coverage;
- Development of a component that simulates an UV Sensor;
- Develop or adapt a coverage path planner;
- Development of planning techniques for disinfection;
- Testing and validation of the techniques developed.

1.3 Thesis Outline

This thesis is organized into six more chapters. Chapter 2 gives an description about the several components needed for autonomous navigation and presents a survey of algorithms used in robotics. Chapter 3 introduces the ROS framework and presents some development tools used on it. After that, a robotic 3D simulator presented. Chapter 4 presents a series of robots developed for disinfecting spaces that use UV sensors to kill pathogens. Then, a brief comparison between the described robots presented Subsequently, two ROS packages developed for coverage path planning are described. Chapter 5 introduces the MiR100 robot used in this project and describes a GitHub repository with useful ROS packages for working with the robot in a simulated environment. Afterwards, the overall architecture of the work is presented and then the system architecture at ROS computation graph level is described followed by the individually explanation of each component used. Chapter 6 presents the results obtained by the system developed. It focus in the tuning of some parameters so the proportion between the coverage of disinfection and the time the robot takes to do it is acceptable. Finally, Chapter 7 discusses those results and suggests improvements that could be done in the future.

CHAPTER 2

Autonomous Navigation

Mobile robot navigation is a field of study that is continually growing in order to substitute humans for repetitive or dangerous tasks. Navigation made by a robot is classified as autonomous when the robot can make automatic judgments based on an evaluation of the surrounding environment. Figure 2.1 shows the components needed by the robot to perform this type of navigation [1]. They are perception, mapping, localization, cognition and motion control.

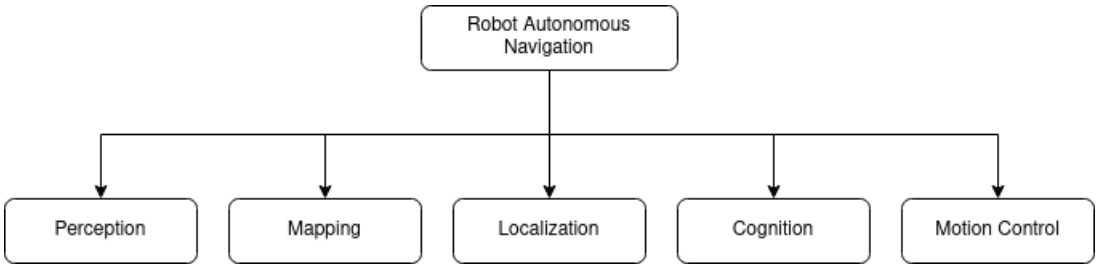

Figure 2.1: Mobile Robot Navigation Blocks

Perception - The perception is the process of extracting the robot's knowledge of its surrounding environment. This is done by extracting meaningful information from the measurements taken by the sensors.

Mapping - Robot mapping is the process of creating a representation of the robot's environment in the form of a map.

Localization - Robot localization is the process of determining the position of a robot in its environment.

Cognition - This is a system's deliberate strategy to achieve its highest-order goals.

Motion Control - This block describes the control of the robot's motor outputs to keep the vehicle on the desired trajectory.

Autonomous mobility is a complex problem, and different constraints must be addressed if the robot must perform in an indoor or outdoor environment. This work focuses on navigation in indoor spaces, so all the techniques mentioned are used in this type of environment.

This chapter will present an explanation alongside some techniques used in localization, mapping, and global and local planning. These five problems must be solved to achieve autonomous navigation.

2.1 Localization

"Where is the robot now?" is the question that needs to be answered when talking about robot localization [2]. Localization can be considered one of the most important blocks regarding robot navigation because, without the robot's estimation of its own position with respect to the environment, all other blocks corresponding to navigation will not receive the correct information about the robot's pose, leading them to provide unreliable data.

When dealing with a robot's position, it is important to determine the reference with which it is associated because that will be critical during the cognition process in Figure 2.1. Because of this correlation, localization entails more than merely detecting an absolute position in space; it entails obtaining a representation of the environment (mapping) and then calculating the robot's position in relation to that map [1].

The use of data from the robot's sensors is critical when attempting to estimate its posture; yet no instrument is completely precise due to noise and aliasing. These two characteristics are well described on [1]. There are mainly three types of localization problems that need to be addressed before proceeding to the localization methods [1][3], being these:

Position tracking - In position tracking, the data provided by the robot's sensors, its previous pose and its odometry is used to monitor the robot's navigation path over time. The initial position of the robot is known.

Global localization - Global localization consists of estimating the robot's pose in relation to the environment.

Kidnapped robot - This problem occurs when the robot thinks it knows where it is but in fact that it is not its correct position. Recovery from this problem is necessary for autonomous navigation.

These three problems must be considered when the robot computes its pose (position and orientation). The robot's localization can be obtained using several methods, such as Markov and Kalman Filter (KF) localization [4][5] which are based on Bayes Filter [6]. The difference between them is that the first uses an arbitrary probability density function over the localization model, whereas the second uses Gaussians to represent the prediction of the robot's position, the localization model, and the information model from the sensors [1]. Monte Carlo Localization (MCL) [7][8] is an extremely popular algorithm used in robotics to solve localization problem. This method is also based on the Bayes filter, but it uses particle filter and can approximate most distributions of practical importance, which gives it an advantage

over the previously mentioned algorithms [9]. Another solution is called Scan Matching, which consists of comparing two consecutive scans of the environment, or the current scan against the map of the environment, and finding the best alignment between them. This alignment is used to estimate the robot's motion and position. This technique can be used to improve the accuracy of the robot's localization, especially in environments where odometry is not reliable [10]. A recent approach to mobile robot localization using scan matching was developed on Laboratory Intelligent Robotics and Intelligent Systems (IRIS Lab) of IEETA. The likelihood field is used as a connection link between scan matching and robotic localization, it avoids the need to establish direct correspondences and reduces computational complexity. The scan matching problem is formulated as a non-linear least squares problem and solved by using the Gauss-Newton and Levenberg-Marquardt methods. Additionally, the authors propose a loss function to reduce the influence of outliers during optimization [11].

Global Navigation Satellite System (GNSS) is one of the most known localization systems for outdoor environments, but it cannot be used when an accuracy of some meters impacts the robot's application. If there was a GNSS's signal in an indoor environment, it would be inadequate to guarantee a useful location of the robot [1][12].

Two probabilistic approaches used nowadays to estimate the robot's actual position in relation to the map environment will be described below.

2.1.1 Adaptive Monte Carlo Localization

Adaptive Monte Carlo Locatization (AMCL) method is an improvement of the MCL algorithm. AMCL recurs to the KLD-Sampling [13], which uses the Kullback-Leibler distance (KLD) to measure the approximation error caused by the sample-based representation of the particle filter. This approach is more efficient since the size of the sample sets (number of particles) is constantly adapted. This technique uses a probabilistic motion model and a probabilistic measurement model to estimate the pose of a robot as it moves through an environment and to estimate the state of the environment or other objects in the environment based on sensor measurements, respectively.

There are several sampling motion models available. Some use control inputs as translational and angular velocities executed over a fixed time interval. However, this type of model requires the addition of a third noise parameter to generate a space-filling posterior probability. Others use odometry readings as inputs, assuming that initial and final rotations and translations are subject to noise. Although technically not controls, using odometry readings similarly to controls simplifies the estimation problem. When it comes to measurement models, beam models of range finders, likelihood fields for range finders, correlation-based models and feature-based models are some of those and more information about them can be find in chapter 6 of [9].

The procedure of AMCL can be described by the Algorithm 1 in which the previous sample set (χ_{t-1}), the map (m), the most recent odometry's control (u_t) and measurement of the sensors (z_t), and the error bounds of the difference between the true posterior and the sample-based approximation (ε) and of the number of samples probability (1-δ) are used as

input. It starts by setting all the bins (b) of the histogram (H) to empty. Then, the algorithm generates new particles while the number of particles in a set (M) does not exceed (M_χ) or a user-defined minimum ($M_{\chi_{min}}$).

The generation of a particle is composed of the two following steps: Firstly, a particle (x_{t-1}) is extracted from χ_{t-1} and as well, as in the MCL algorithm, a new particle ($x_t^{[M]}$) is predicted based on one of the motion models described, then the importance weight ($w_t^{[M]}$) of $x_t^{[M]}$ is calculated using one of the measurement models described, and finally the particle is inserted into the new sample set (χ_t). Secondly, the KLD-sampling is implemented, and when a $x_t^{[M]}$ falls into an empty b, the number of non-empty histogram bins k increases. After that, the values k, δ, and ε are used to compute the number of particles in the statistical bound M_χ.

Algorithm 1: KLD-Sampling-MCL Algorithm [9]

Input: Previous weighted sample set - χ_{t-1}
Input: Map - m
Input: Most recent control - u_t
Input: Most recent measurement - z_t
Input: Statistical error bound between the true posterior and the sample-based approximation - ε
Input: Statistical error bound of the sample-based approximation quality - δ
Output: Actual weighted sample set - χ_t

Function `AMCL`$(\chi_{t-1}, u_t, z_t, \varepsilon, \delta)$
  $\chi_t = \{\ \}$;
  $M = 0$, $M_\chi = 0$, $k = 0$;
  foreach *(b in H)* **do**
    b = empty;
  end
  do
    draw i with probability $\propto w_{t-1}^{[i]}$;
    $x_t^{[M]}$ = sample_motion_model$(u_t, x_{t-1}^{[i]})$;
    $w_t^{[M]}$ = measurement_model$(z_t, x_t^{[M]}, m)$;
    $\chi_t = \chi_t + \langle x_t^{[M]}, w_t^{[M]} \rangle$
    if *($x_t^{[M]}$ falls into empty bin b)* **then**
      $k = k + 1$;
      b = non-empty;
      if *($k > 1$)* **then**
        $M_\chi := \frac{k-1}{2\varepsilon} \left\{ 1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}} z_{1-\delta} \right\}^3$;
      end
    end
    $M = M + 1$;
  while *($(M < M_\chi)$ // $(M < M_{\chi_{min}})$)*;
  return χ_t;
End

2.1.2 Scan Matching-based Localization

Scan Matching method determines the relative translation and rotation displacement of the robot's pose by overlapping the current map relative to the pose of the robot on successive sets of raw range readings, or against a map [14][15][16]. This technique offers high precision and computational efficiency. Iteractive Corresponding Point (ICP), introduced in [17], and its variants in [18][19] are used to solve the minimization problem generated by the method used for point registration. Pedrosa *et al.* [11] and Burguera *et al.* [20] use a technique called likelihood field as measurement model, which is a variant of ICP where the direct association of the data does not exist turning the process more efficient.

2.2 Mapping

In mobile robot navigation, the ability to model the environment where it is moving is essential. Robots must have sensors that allow them to perceive the outside environment to obtain a map. Sensors' measurements are subjected to noise, which, together with the range limitations of the instruments, are the main problems of mapping.

Due to the existence of diverse environments and mapping' methods, the maps can be represented in a 2-D grid, a 2.5-D grid, or a 3-D grid. The focus of this study is indoor mapping, and for that reason, only 2-D grid approaches will be considered. That does not mean that 2.5-D grid and 3-D grid are not used for indoor mapping, only that these representations of maps require more computational resources, and when the criterion of choice is a trade-off between efficiency and accuracy, these ones are far less used.

One of the representations is called Line Maps. Since indoor environments contain primarily linear structures such as lines and planes, this type of mapping is efficient and accurate. Data association as well as knowing "how many lines are there" are the principal problems of this technique and algorithms such as split-and-merge [21], expectation maximization (EM) [22], RANSAC [23], Hough-Transform [24] and Line-Tracking [25] were developed to deal with those. An implementation of this method can be seen in [26][27][28].

Another category of environment description is Topological Maps. In this method, the map is represented by a graph-like structure in which significant places are pointed out and attached via arcs. An initial approach to this technique can be seen in [29], later generalized Voronoi graphs (GVG) started to be used to obtain the roadmap of the environment. Choset and Nagatani [30] employ GVG, via a graph matching process, to obtain the localization of a robot on a partially explored map. Cheng et al. [31] developed a system that generates a topological map and uses it together with the loop closing method to build the environment map through autonomous exploration. In [32], the original GVG is pruned in order to get a more compact and efficient topological mapping. They have managed to produce an algorithm that is shorter by several orders of magnitude in terms of total length than other state-of-the-art methods and is at least 30% faster than them.

Landmark-based Maps are another possibility. This type of mapping is promising in environments with distinguishable features. Knowing the robot's pose and estimating the

position of the individual landmarks is the goal of this approach [33]. Murrieta-Cid et al. [34] use a landmark-based model to develop a visual navigation system in natural environments.

In this section another technique will be explained, the occupancy grid method, which can be considered the most popular when talking about mapping on mobile autonomous navigation.

2.2.1 Occupancy Grid

This type of mapping model is also known as Grid Maps, and it is an extremely popular way of representing the environment. As mentioned in [35], this method is known for being exceptionally robust and easy to implement.

Occupancy grid maps represent the environment by a grid of cells, where for each cell, information about the area it covers is stored in the form of a single value corresponding to the state of being occupied or unoccupied [36][37]. This approach is solved by the generation of probabilistic maps, where the algorithm used to define the occupancy of a cell is based on Bayes filter described in [33].

The works of Thrun [35] and Stachniss [37] mention some disadvantages of using occupancy grid maps such as the nonexistence of a technique for accommodating pose uncertainty, the assumption of independent noise made by the Bayes filter, the existence of discretization errors, and the requisition of a lot of computational resources.

2.3 Global Path Planner

In autonomous navigation, a Global Planner is an algorithm that creates a path from a starting position to another position on a given map. This method recurs to path-finding algorithms as well as costmaps to identify the most optimal path. One drawback of this approach is the need to use the whole map to compute the path, which requires a large computational load. Path-finding algorithms are used to compute the less costly path from one point to another. There are several techniques where each one can use a different method to calculate the distance between the points that corresponds to the cost of navigation.

Dijkstra's algorithm [38] is widely used in a weighted grid with non-negative edge weights, which means that in this method, the map is represented by a grid in which each edge has a cost or weight assigned to it. This weight symbolizes some aspect of the edge, such as traversal time, cost, or distance. Starting from the source node, the algorithm explores the grid breadth-first where the cells are visited in a first-in-first-out (FIFO) way like the figure 2.2 describes, constructing a shortest-path tree as it goes.

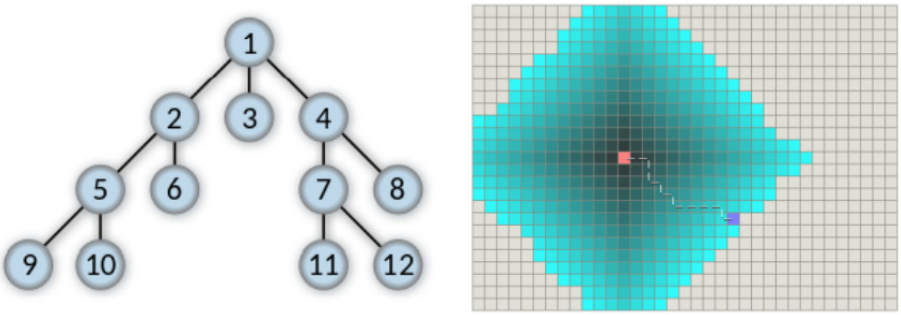

Figure 2.2: Breadth-first search planner (left) and Example of task performed by Dijkstra's algorithm (right) [39].

Other common algorithms used to obtain the shortest path are A-Star and Search-based Planning Lattice.

2.3.1 A-Star Algorithm

The A-Star Algorithm (A*) [40][41] is a method based on Dijkstra's algorithm that searches by using a heuristic function to guide the search towards the goal node (Figure 2.3). The heuristic function aids in prioritizing nodes that are more likely to lead to the objective, improving the runtime and memory consumption efficiency of the A* algorithm.

In grid-based path-finding problems, a common heuristic function is the Manhattan distance, which is the sum of the absolute differences in the `x` and `y` coordinates between two cells. Other common heuristic functions include the Euclidean distance and the Chebyshev distance. The Euclidean distance is the straight-line distance between two points, while the Chebyshev distance is the maximum of the absolute differences in the `x` and `y` coordinates between two points. Algorithm 2 is a basic description of this technique.

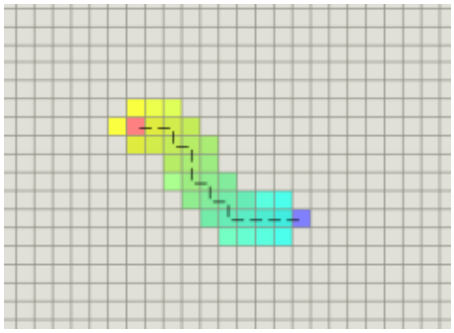

Figure 2.3: Path from the orange point to the purple point generated by A* algorithm [39].

While efficient, A* seeks an optimal path, which may not be possible given time constraints and the size of the environments in which autonomous vehicles must operate. Anytime variants of A* search, such as Anytime Reparing A-Star Algorithm (ARA*) [42] and Anytime Dynamic A-Star Algorithm (ADA*) [43], have been developed to deal with limited deliberation time. These algorithms quickly generate an initial, potentially suboptimal solution and then focus on improving it while deliberation time allows.

Algorithm 2: Pseudo-code to perform basic A* [39]

```
Input: Start/Goal Configurations - start, goal
Input: Costmap information
Output: Planned path totalpath or failure indication
Function aStar(start,goal)
    closedset ← { };
    openset ← {start};
    origset ← { };
    gscore[start] ← 0;
    fscore[start] ← gscore[start] + heuristic(start, goal);
    while (openset is not empty) do
        current ← node in openset with lowest fscore;
        if (current = goal) then
            totalpath ← [current];
            while (current in origset) do
                current ← origset[current];
                totalpath.append(current);
            end
            return totalpath;
        end
        remove current from openset;
        add current to closedset;
        for (each neighbor in neighbor_nodes(current)) do
            if (neighbor in closedset) then
                continue;
            end
            temp_gscore ← gscore[current] + distance(current, neighbor);
            if ((neighbor not in openset) || (temp_gscore < gscore[neighbor])) then
                origset[neighbor] ← current;
                gscore[neighbor] ← temp_gscore;
                fscore[neighbor] ← gscore[neighbor] + heuristic(neighbor, goal);
                if (neighbor not in openset) then
                    add neighbor to openset;
                end
            end
        end
    end
    return failure;
End
```

2.3.2 Search-based Planning Lattice

The Search-based Planning Lattice (SBPL) global planner was created to tackle the restrictions imposed by methods that apply efficient graph searches or use highly informative heuristics to guide the search for feasible paths on a systematic discretization of the environment. This type of method involves an intensive computational load when applied over large distances, and SBPL aims to compute complex feasible plans over this environment's conditions [44].

SBPL utilizes the work done in [45], which presents a method to create an efficient search space for constrained motion planning by encoding only feasible motion plans and using a systematic approach to generate a minimal set of distinct motion alternatives to construct its multi-resolution lattice state space that is used along with an incremental search for planning the suboptimal path in the lattice. A state lattice is a discretization of the configuration space into a set of states representing configurations and connections between these states, where each connection indicates a possible path. An example of a lattice can be seen in Figure 2.4.

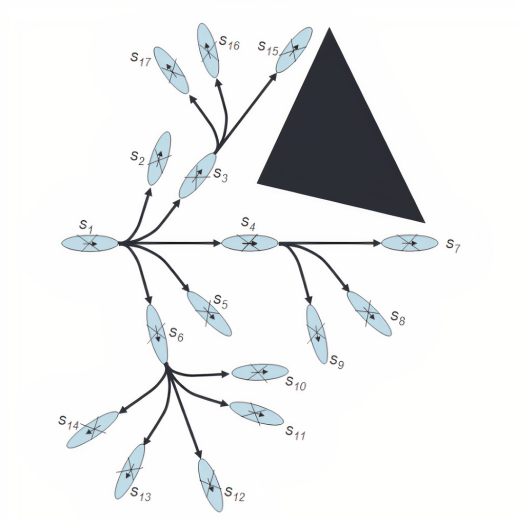

Figure 2.4: A 3D (x, y, θ) lattice with at most five forward actions for each state and no backward actions. A full set of actions is shown for states s_1. For every state, this set of actions is translated and rotated appropriately, and all actions intersecting obstacles are removed [44].

Multi-resolution lattice state space:

The construction of a lattice needs to consider the state space and the action space. The first one refers to the sampling strategy used for representing the states in the lattice, and the second is the control set used for inter-state connections [44]. The state space is represented by a four-dimensional state, (x, y, θ, υ), where (x, y) corresponds to the position of the center of the vehicle in the world, θ is the orientation of the vehicle, and υ is the translational velocity of the vehicle.

An action space can be obtained by the following steps. Firstly, the algorithm computes a group of feasible actions using the trajectory generation algorithm described in article [46] and the present state of the vehicle. This method has the advantage that each action produced has its endpoint landing on a lattice state. Secondly, a subset of states is calculated. These states need to be reachable via a feasible action within a distance between the states of the present state. Thirdly, the group of feasible actions is checked to see if some longer feasible action can be represented by a set of shorter feasible actions. If that is the case, the longer feasible action is removed from the group. Finally, the compact action space that represents the entire reachable space from the present state is obtained. Figure 2.5 left side image shows the action space in the lattice for a single state.

The multi-resolution lattice approach is implemented to tackle the problem of high computation and memory usage. This technique uses a high-resolution action space (Figure 2.5 left image) in the neighborhood of the robot and the goal, and a low-resolution action space (Figure 2.5 right image) elsewhere. The key is to ensure that the high-resolution and low-resolution lattices join nicely.

Likhachev and Ferguson [44] described the multi-resolution lattice method where, the action space used in the low-resolution space is a subset of the action space used in the high-resolution space. In terms of a high-resolution discretization and a lower-resolution discretization of variables (x, y, θ, v), the building of the action space used in the low-resolution space can be described mathematically as follows. A feasible action that connects two states only belongs to the action space of low-resolution space if that same feasible action belongs to the high-resolution space and at the same time to the low-resolution discretization. Figure 2.5 right side shows us an image of a low-resolution action space obtained from the high-resolution action space of Figure 2.5 left side image.

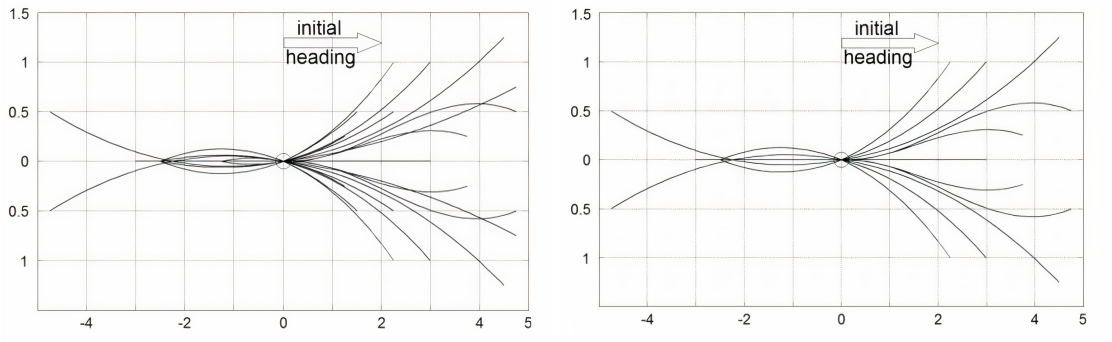

Figure 2.5: Example of a High-resolution action spaces (Left) and a Low-resolution action spaces (Right) [44].

Incremental search:

Now that the multi-resolution lattice is constructed, the ADA* algorithm mentioned in Section 2.3.1 is used to perform a backwards incremental search from the goal configuration towards the current configuration of the vehicle to plan and re-plan the robot's global path while the vehicle moves.

The heuristic used on the ADA* is a single Dijkstra's search that is redone every time the vehicle pose changes. This search is done until the cost of the cell it is about to calculate is greater than or equal to the cost of the goal cell. A cell to be searched needs to be reachable by an existing state that is no more than twice as far, in terms of path cost, than the cell the center of the robot is in. Path cost refers to an optimal solution through the search space assuming a perfectly empty environment, which is a highly effective general heuristic. The search starts at the robot's pose cell, and the costs of the shortest pathways from that pose to all the other cells in the surroundings that obey the criteria previously mentioned are calculated.

Optimization:

Calculating the cost of actions in navigation planning is one of the most computacionally costly aspects because it requires convolving the geometric footprint of the vehicle for a specific action with a perception map. This problem is optimized using two steps. Firstly, when the robot performs an action, we precomputed the cells covered by the vehicle for that individual action. Then, we create two configuration space maps that the planner can use to avoid performing convolutions. The first of these maps, using the robot's inner radius as a reference, it expands all the obstacles in the perception map. The second map uses the outer radius as reference to do the same as the first map.

Iterating over the states that may possibly be affected by changes in the costmap is a process that, like the calculation of the cost of actions, requires a certain amount of computational effort. To mitigate this, the ADA* algorithm must only update the values of the states that have been calculated in prior planning iterations. So, there is no need to update the value of a state if it has not been calculated [44].

2.4 Mapping Coverage

Aside from global path planners, coverage algorithms are an important component of robotics and autonomous systems. Mapping coverage consists of the ability to determining a path that goes over all the points of an area of interest while taking into account different requirements, such as shortest path, fastest path, percentage of area covered or the robot making a specific movement. In a map, a global planner algorithm is used to plan a route from the beginning point to the destination. It considers the surroundings, obstructions, and other criteria to select the best route. To discover the shortest path, global planners often rely on maps of the environment and techniques such as A* or Dijkstra's algorithms.

A map coverage algorithm, on the other hand, is used to explore and cover a complete area rather than determining a specific path from one point to another. Map coverage algorithms function by breaking down the area to be covered into smaller parts or cells and then determining which cells must be covered and in what sequence. They can employ several strategies, such as spiral spanning tree coverage or random coverage, to ensure that the entire region is covered.

In essence, global planner algorithms and map coverage algorithms differ in that global planners seek the shortest path from a starting point to a destination, whereas map coverage algorithms seek to cover a whole area taking into account different criteria as aforementioned. While both types of algorithms are vital for autonomous navigation, they serve different purposes and use different strategies to accomplish their objectives.

2.4.1 Spiral Spanning Tree Coverage

Spiral Spanning Tree Coverage (Spiral-STC) is an online sensor-based algorithm that uses an in-robot square-shaped tool to cover a space. This technique assumes that the covering tool is a square of size D, the robot is only allowed to move the tool in orthogonal directions

to the tool's four sides without rotating the tool during this motion, and the environment is approximated by a discrete grid of D-size cells.

The initial version of this algorithm, $2D$-Spiral-STC, subdivides the work area into $2D$-size cells where only the completely free ones are used. This condition makes it impossible to cover all of the map's area. A detailed explanation can be found at [47]. A later version of Spiral-STC [48][49], called Full-Spiral-STC, subdivides the $2D$-size cells into D-size subcells. This approach makes it necessary to consider the different types of edges (figure 2.6). The following procedure obtains the trajectory between free cells: Being `x` the current cell and `y` a neighbor cell. The covering tool moves from its current subcell in `x` to a subcell of `y` by following the right side of the spanning tree edges, measured concerning the tool's direction of motion.

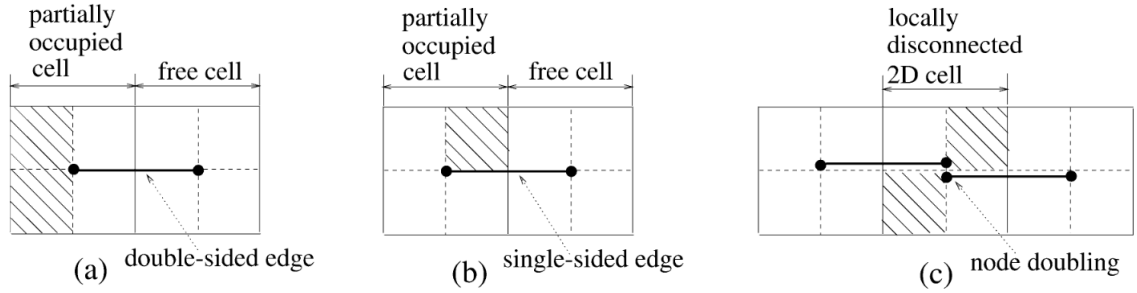

Figure 2.6: Types of subdivision of a 2D-size cell into a D-size subcell. (a) A double-sided edge; (b) a single-sided edge; (c) node doubling at a disconnected cell. [48]

The existence of partially occupied edges requires a specific motion: Being α the path that would have been taken if the cell were completely free, and β the actual path taken by the covering tool. The trajectory consists of deforming α away from the occupied subcell without changing the path's direction until every point of β lies at a distance of $D/2$ from the occupied subcell. Figure 2.7 shows a representation of these two types of motion.

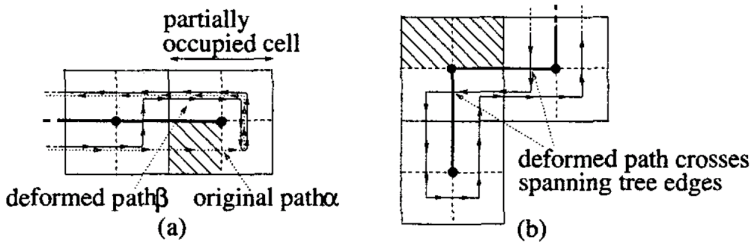

Figure 2.7: (a) Path deformation along a single-sided edge; (b) Crossing of spanning-tree edges. [48]

Algorithm 3 describes the steps of the Full-Spiral-STC procedure where a free cell is considered `new` if all of its subcells are not yet covered [48]. The recursive method receives `S` and the parent cell `w` as inputs and starts by changing the status of actual cell `x` to `old`. Then, starting with `w`, the algorithm searches for a `new` cell neighbor of `x` in counter-clockwise order and assigns it to `y` (`scanNewNeighbour(w,x)`). The next step is to generate a spanning tree between `x` and `y` (`generateST(x,y)`). After that, use it alongside the path

constructed to move from `x` to `y` taking into account the type of edges between these two cells (`moveToSubcell(...)`). This process is repeated until the current cell does not have a `new` free or partially occupied cell, but in the following interaction inputs, `x` will be the previous `y` and `w` the previous `x`. Finally, the robot returns to the `S` cell. An application of this algorithm can be seen in figure 2.8.

Algorithm 3: Full Spiral-STC Algorithm

```
Data: Call STC2(Null,S)
Data: Starting cell - S
Input: Parent cell - w
Input: Actual cell - x
Output: Returns the path for the full coverage of the grid map
1  Function STC2(w, x)
2      x = old;
3      while (x has new free or partially occupied neighboring cell) do
4          cell y = scanNewNeighbour(w,x);
5          spanningTree st = generateST(x,y);
6          moveToSubcell(x,y,st);
7          STC2(x,y);
8      end
9      if (x != S) then
10         moveToSubcell(x,w,Null);
11     end
12 End
```

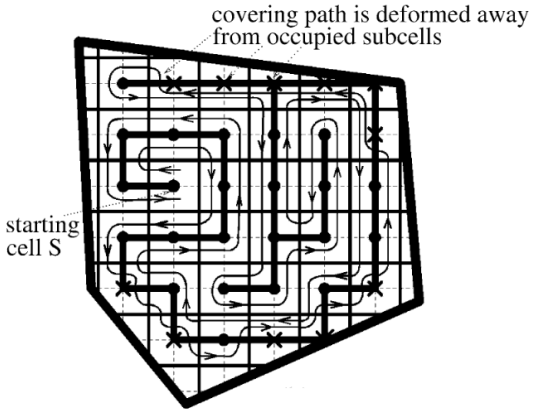

Figure 2.8: Full Spiral-STC Coverage. The figure possesses two unrealistic features, which were added for clarity. The tool size D is shown unrealistically large with respect to the work area size, and the covering tool path is shown curved while it is rectilinear according to the algorithm. [48]

2.4.2 Backtracking Spiral Algorithm

The Backtracking Spiral Algorithm (BSA) is a mobile robot coverage strategy that employs spiral filling paths in conjunction with a wall-follower algorithm [50][51]. Unvisited regions

are marked and covered by a backtracking mechanism to ensure completeness. The switch between the modified grid-based spiral path functionality and the wall-following procedure is made according to the detection and visitation of unvisited obstacles.

The spiral path algorithm is a set of conditions that define the robot's route from the region's boundary to a central ending point. In this method, the Reference Lateral Side (RLS) indicates the side where obstacles are to be referenced and Opposite Lateral Side (OLS) the opposite direction of RLS. While executing this method, the algorithm continuously searches for unvisited obstacles. If any are detected, the wall-following component is activated to circumnavigate the obstacle. The wall-following algorithm aligns the obstacle on the RLS of the robot, stores its first position and starts circumnavigating the object until it reaches the initial location and starts the spiral path component. During this procedure, the algorithm detects and marks all the virtual obstacles and Backtracking Points (BPs).

The backtracking method (Algorithm 4) is used to return to previously unvisited areas where a new spiral-filling procedure can be performed. During the execution of a spiral path, BPs, which are the cells where there is more than one unknown cell in its vicinity, are detected and saved before the robot moves forward.

Algorithm 4: Backtracking Method

```
Data: Backtracking Point - bp
Data: Candidate Backtracking Point - candidateBP
Input: List of Backtracking Points - bps
Input: Criteria Method to Choose the Candidate Backtracking Point - criteria
Output: Returns the robot's path to the candidate cell
1 Function BactrackingMethod(bps,criteria)
2     forall (bp in bps) do
3         if (isNotUnkownCell(bp)) then
4             discard(bp);
5         end
6     end
7     candidateBP = selectCandidate(bps,criteira);
8     pathCrossing(candidateBP);
9 End
```

The function `isNotUnkownCell(bp)` checks if the BP is marked on the grid-map as unknown. `discard(bp)` removes that BP from the BPs list. `selectCandidate(bps, criteria)` chooses using the criteria method provided (e.g. euclidean distance between the BP and the robot) the BP to where the robot goes. Finally, `pathCrossing(candidateBP)` returns the robot's path to the candidate cell. This method considers the already free and partially-occupied cells as candidates, attributing a cost to each possible path found. The path's length and paths mainly composed of free cells are two criteria used to decide which route the robot will use. After selecting the return path, the Virtual Pipe approach (Algorithm 5) is used to guide the robot throw that track from the ending spiral point to the selected BP. The procedure starts by defining a reference line between all the central points of the chosen path (`referenceLine(path)`) and computing the virtual pipe area using this

reference line (`virtualPipeArea(ref)`). Figure 2.9 shows an example of the described steps. Then, the robot moves along this reference line until it reaches the candidate BP (`reachFinalRefPoint(candidateBP)`). During this process, if the cell on the reference line is partially occupied (`obstructedCell(ref.point)`), then the robot will check if its center exits the virtual pipe area while circumnavigating the obstacle on its RLS (`exit(pipeArea, RLS)`). If this condition is present, the wall-following method is performed with the obstacle at the OLS of the robot (`wallFollowing(OLS)`). The condition being negative, the same technique is used with an obstacle at the RLS of the robot (`wallFollowing(RLS)`). The circumnavigation of the barrier stops when the robot returns to a cell of the reference line (`cellsOnRefLine()`).

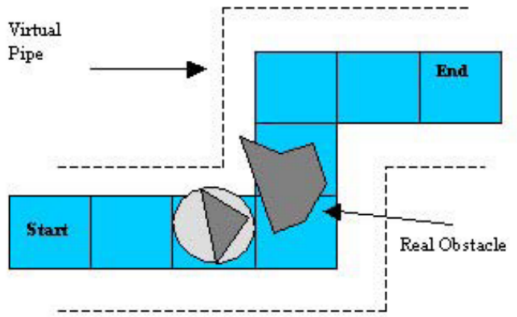

Figure 2.9: Example of a virtual pipe [51].

Algorithm 5: Virtual Pipe Algorithm

```
Data: Cell on the reference line - ref.point
Data: Candidate Backtracking Point - candidateBP
Data: Virtual Pipe Area - pipeArea
Data: Side where obstacles are to be reference - RLS
Data: Opposite direction of RLS - OLS
Data: Reference line between all the central points of the input path - ref
Input: Path returned from the Backtracking Method - path
Function VirtualPipe(path)
    ref = referenceLine(path);
    pipeArea = virtualPipeArea(ref);
    while (!reachFinalRefPoint(candidateBP)) do
        if (obstructedCell(ref.point)) then
            if (!exit(pipeArea,RLS)) then
                while (!cellIsOnRefLine()) do
                    wallFollowing(OLS);
                end
            end
            else
                while (!cellIsOnRefLine()) do
                    wallFollowing(RLS);
                end
            end
        end
        else
            moveTo(ref.point.next());
        end
    end
End
```

2.5 Local Path Planner

The local planner recurs to a local costmap to keep the robot on the path generated by the global planner algorithm. Its principal function is collision avoidance, so computation speed is a must. To acquire such characteristics, this algorithm uses only a tiny portion of the environment map that corresponds to the surrounding area of the robot and builds a costmap of that region.

This section will describe the Dynamic Window Approach (DWA) and Timed Elastic Band (TEB) methods used in local planning.

2.5.1 Dynamic Window Approach

The restrictions imposed by velocities and accelerations led to the creation of DWA. This technique is based on the motion dynamics of synchro-drive mobile robots. It uses only a short time interval when calculating the following steering command to eliminate the enormous complexity of the general motion planning problem [52].

The DWA algorithm 6 is subdivided into two steps. Firstly, the search space phase is subdivided into three parts: the circular trajectories where the robot obtains all the curvatures forming a two-dimensional velocity search space; the admissible velocities where the previous search space is restricted to those curvature values in which it is possible to stop the robot before it reaches the closest obstacle; and the dynamic window where the remaining search space is reduced to the velocities that the vehicle can reach in a short time interval (`X`) given the robot's limited accelerations. Secondly, the optimization phase in which an objective function σ in algorithm 6 is maximized. A trade-off between the progress towards the goal location (`heading()`), the distance to the closest obstacle on the trajectory (`dist()`), and the forward velocity of the robot is made concerning the current pose (position + orientation) of the robot.

Algorithm 6: DWA Algorithm

Data: Translational Velocity - υ
Data: Rotational Velocity - ω
Data: Type curvature represents the set of velocities (υ, ω)
Data: Short time interval - `X`
Input: Weight correspondent to the target heading - α
Input: Weight correspondent to the clearance - β
Input: Weight correspondent to the velocity - γ
Output: Velocity of search space V_r with highest value of maximization

```
1  Function DWA(α, β, ϒ)
2      curvature[ ] V_r, V_s, V_a, V_d;                    /* Circular Trajectories */
3      V_s = getAllCurvatures();
4      curvature curv;
5      foreach (curv in V_s) do                            /* Admissible Velocities */
6          if (canStopBeforeReachClosestObstacle(curv)) then
7              V_a ←curv;
8          end
9      end
10     foreach (curv in V_a) do                                  /* Dynamic Window */
11         if (velReachedWithinTimeFrameX(curv,X)) then
12             V_d ←curv;
13         end
14     end
15     V_r = V_s ∩ V_a ∩ V_d;
16     curvature[ ] G;                                            /* Optimization */
17     foreach (curv in V_r) do
18         G ← σ(α∗heading(curv) +β∗dist(curv) +γ∗vel(curv));
19     end
20     return maxValue(G);
21 End
```

2.5.2 Time Elastic Band

The Time Elastic Band (TEB) [53][54] recurs to the deformation of the path created by the global planner in order to avoid obstacles. This method is an improvement of the

Elastic Band (EB) planner [55] because it uses the robot's poses, as EB, alongside a temporal component that allows it to control the speed and acceleration of the robot's motors.

The planner applies the techniques of external repulsion and internal contraction. Detecting obstacles triggers the first one, causing the path to deviate from those points. The second is activated by the generated specific points that cause the computed path to converge in the direction of the global planners' path. The combination of these two types of deformations allows it to avoid obstacles through repulsion and faithfully follow the global plan through contraction. Figure 2.10 illustrates the behavior of the algorithm when encountering obstacles along the path.

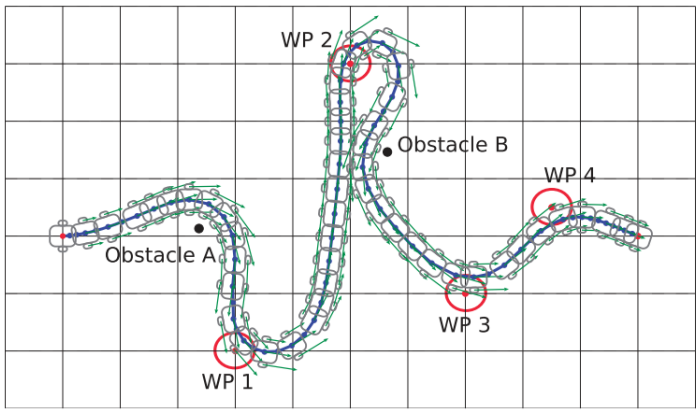

Figure 2.10: Large scenario with consideration of way-points and obstacles. [54]

In TEB, the repulsion method expresses collision avoidance as an objective function in terms of piece-wise continuous, differentiable cost function. These cost functions penalize the robot for violating the collision limitation, repelling it away from obstacles. The contraction method is used for trajectory optimization by iteratively deforming the initial path generated by the global planner, considering objectives such as path length, execution time, obstacle avoidance and compliance with motion constraints. This technique uses temporal information to be able to consider dynamic constraints as well as real-time adaptation of the trajectory [54].

2.6 The Localization and Mapping package

The Localization and Mapping (LaMa) package is an advanced software library created at IRIS Lab in the University of Aveiro. Focused on robotic localization and mapping, the package offers solutions to complex problems in this field. Here's a breakdown of the content and features provided by the package:

Sparse-Dense Mapping Framework

Traditional volumetric grids used in mapping take up a lot of memory. This problem is addressed by LaMa, which introduces a sparse-dense data structure that significantly improves

space and memory efficiency [56]. In addition, an online data compression strategy is employed, which improves space efficiency without sacrificing time efficiency.

Localization based on Scan Matching

This approach employs a likelihood field and non-linear least squares optimization techniques for efficient robot localization [11]. By iteratively minimizing scan matching errors, it accurately determines the transformation between current scans and a reference map. The accuracy of this method is validated by comparing results with ground-truth data, showcasing reduced iterations and high computational efficiency.

Online SLAM with Dynamic Likelihood Field

In this technique is introduced a rapid scan matching approach for online SLAM [57]. It leverages a dynamic likelihood field, establishing connections between scan matching and online SLAM without the need for direct correspondences.

Multi-threaded Particle Filter SLAM

A Rao-Blackwellized particle filter is employed here [58]. This technique utilizes particles to approximate the posterior distribution, updating both the map and the robot's pose. Crucially, the library incorporates resampling methods to maintain particle diversity and prevent depletion. Additionally, likelihood smoothing and adaptive resampling techniques are employed to optimize the filter's performance.

In essence, LaMa stands as a robust and innovative toolkit. Its contributions in sparse-dense mapping, efficient scan matching, and advanced particle filter SLAM techniques make it a really good resource.

CHAPTER 3

Robot Operating System

Developing software for robotic applications from scratch is a challenging task. Due to the huge range of hardware and data that robots rely on, even the most basic applications frequently require extremely sophisticated programming. The Robot Operating System (ROS) framework has as its official definition the following description:

> "ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers."
>
> *http://wiki.ros.org/ROS/Introduction*

ROS addresses this issue by serving as a general-purpose robotics framework. It is not an operating system, despite its name, but rather a type of middleware that works as a scheduler, loader, monitor and error handler between applications and distributed computing resources (Figure 3.1).

Figure 3.1: ROS as a Meta-Operating System [59].

At the communication level, ROS processes form a peer-to-peer network that processes data together. This description only applies to ROS version 1.There is a ROS version 2 with different characteristics, particularly regarding the Master, but it will not be addressed. There are seven concepts that we need to understand to be able to master this framework[1]:

Nodes - ROS is designed to be a decentralized system where nodes are the processes that perform computation. ROS nodes attempt to build small processes rather than huge processes with all the features. In the case of a single node error, it is easier to debug and will not affect the entire program.

Master - The ROS Master provides name registration and lookups to the rest of the nodes. The Master's job is to allow individual nodes to find one another. Once these nodes have found each other, they communicate between themselves on a peer-to-peer basis.

Messages - The node sends or receives data between nodes via a message. Messages are data structures that represent the information exchanged. Arbitrarily nested structures and arrays can be included.

Topics - An unidirectional communication channel between nodes is based on a publisher/subscriber system. The topic is a term used to identify the message's content that is exchanged in this channel. A node interested in a specific data type will subscribe to the relevant topic.

Services - A bidirectional communication channel between nodes is based on a server/-client system. Services are used when there is a need for a request/response interaction where one node might request another to execute a rapid process.

Parameter Server - The parameter server allows us to maintain data in a centralized location. These values are accessible and modifiable by all nodes. The parameter server is located on ROS Master.

Bags - Bags are a format for storing and retrieving ROS message data. Bags are a crucial method for storing data that might be difficult to acquire but is required for creating and testing algorithms.

Figure 3.2 represents an example of the ROS computation graph level. We can see that a single topic may have numerous concurrent publishers and subscribers, and a single node may publish and/or subscribe to multiple topics. In general, neither publishers nor subscribers are aware of the presence of each other.

[1] http://wiki.ros.org/ROS/Concepts

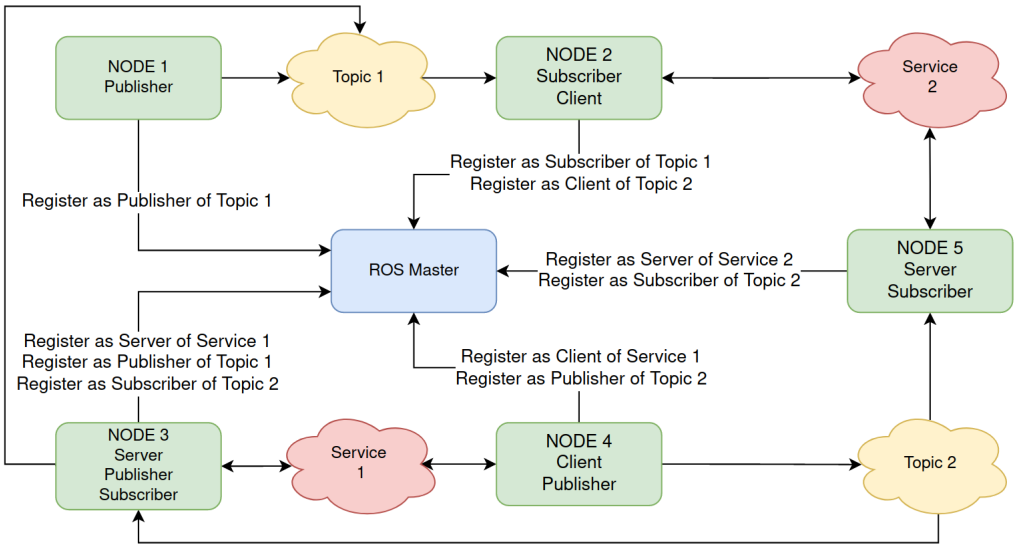

Figure 3.2: Example of ROS Computation Graph Level

We can create a program that does all the computations required in our application, or we can create sub-programs, each with a specialized function, with the latter being typically preferable. ROS is also open source, which means we may utilize it for nearly any purpose we see fit for the development of our program, such as altering pre-existing code for a specific situation.

3.1 Development Tools

When developing a program using ROS, several tools can help us debug. In this section we describe two tools that were used in this project.

3.1.1 RViz

RViz is a useful 3D visualization tool for ROS. There is a ROS package called `rviz`[2], where the node that runs this application is described. The user can subscribe to multiple topics and see how the system interprets the information. Figure 3.3 represents a possible environment when we use this tool.

[2] http://wiki.ros.org/rviz

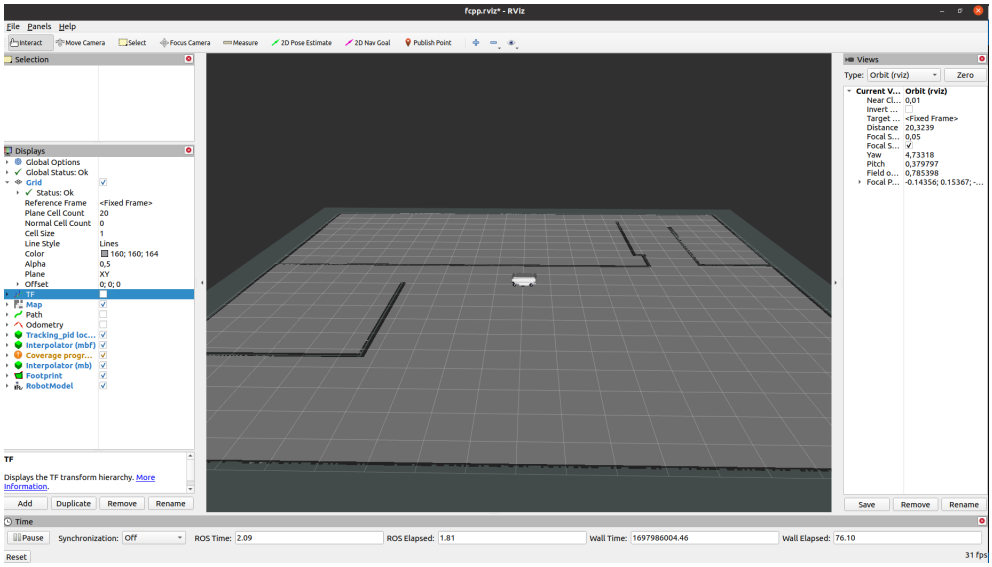

Figure 3.3: RViz

3.1.2 rqt_graph

This application shows us how multiple nodes and topics are related. It generates a graph of all of them and how they interact. Figure 3.4 displays the environment the user sees when using this tool.

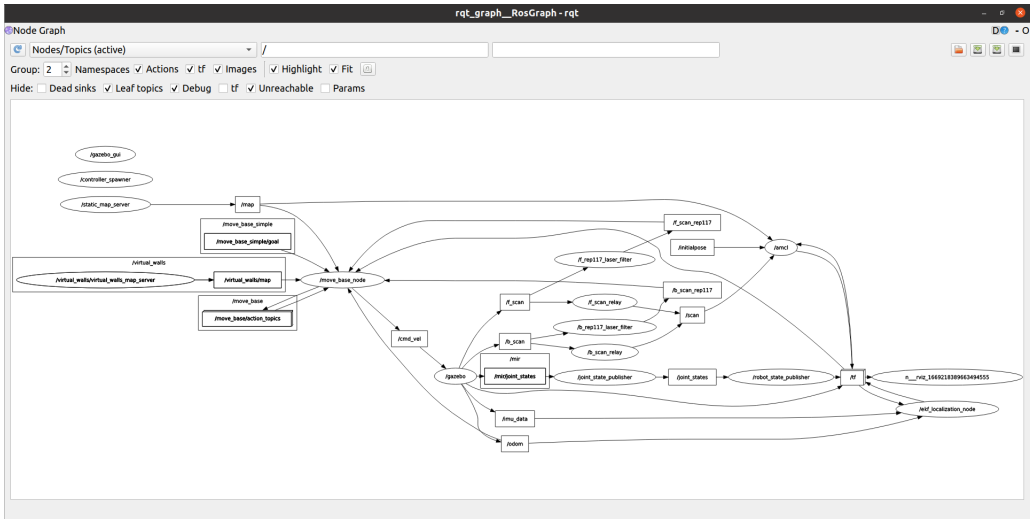

Figure 3.4: rqt_graph

3.2 Gazebo

Gazebo is one of the most popular open-source 3D simulators because it includes robots, sensors, environmental models, and realistic simulations with its physics engine. This framework is a very popular robotics simulator due to its outstanding performance. Figure 3.5 shows the appearance of the gazebo simulator.

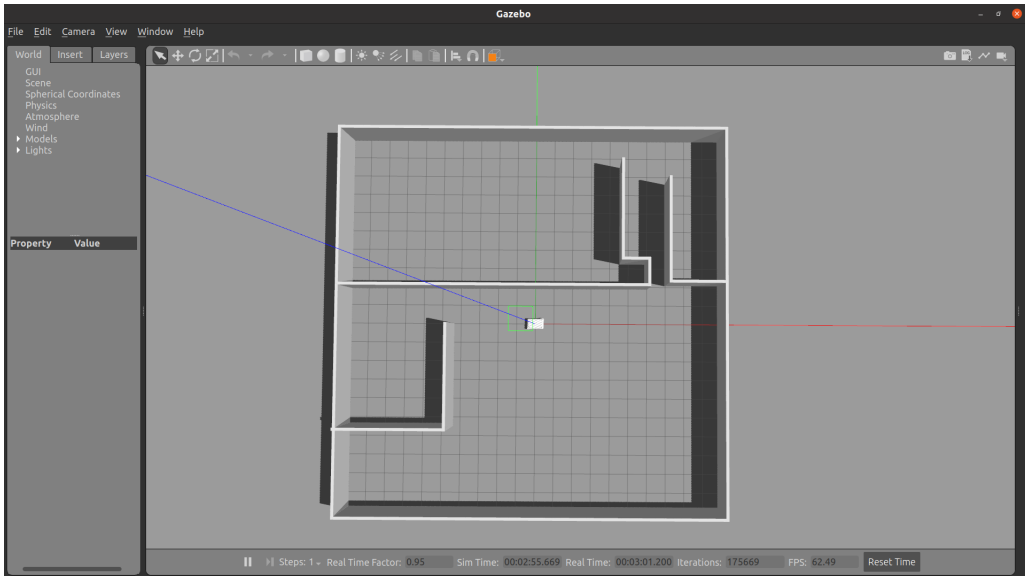

Figure 3.5: Gazebo

Due to being created and provided by Open Robotics, which is responsible for maintaining ROS, this software is compatible with the robot operating system. A ROS package called `gazebo_ros`[3] is available, which offers wrappers, tools and additional APIs for using ROS with the Gazebo simulator.

[3] http://wiki.ros.org/gazebo_ros

CHAPTER 4

Robot Disinfection State of Art

Outbreaks of infectious diseases represent a substantial hazard to human health. It is believed that the frequency of such breakouts has grown during the last decade. Some microorganisms that were not previously thought to constitute a general risk to human health have arisen at the regional and global levels [60].

Mass-gathering constructed settings may become hotspots for pathogen colonization, transmission, and exposure, spreading infectious illnesses among people in towns, cities, nations, and throughout the world. Infectious illness epidemics inflict enormous costs on our society. Frequent cleaning and disinfection, as recommended by the World Health Organization (WHO) and the Centers for Disease Control and Prevention (CDC), are crucial for reducing pathogen transmission and exposure and slowing the development of infectious illnesses [61].

The COVID-19 pandemic has been present since late 2019 and has created critical problems in practically every country. Disinfection has begun to be required for all public meeting places, including schools, airports, transportation systems, and hospitals [62]. This process is mainly done by a human workforce, which can be labor-intensive, time-consuming, and harmful to one's health, reducing disinfection's efficacy and efficiency. A virus may persist for extended periods on several surfaces and spread swiftly within constructed habitats [61].

Robots are an excellent answer to this problem since the virus cannot multiply inside a robot, and the usage of robots significantly minimizes person-to-person interaction [63]. In the last half decade, the number of work done in robotics has increased in several areas like public safety, clinical care, continuity of work and education, quality of life, laboratory and supply chain automation, and non-hospital care [64]. This work will focus only on the two first areas, more precisely on the disinfection of public spaces and the disinfection point of care.

This chapter begins by presenting some robots currently used to disinfect public spaces. After that, the theme of robotic exploration will be addressed, and two methods for coverage path planning will be explained.

Disinfection robots can use two methods to "kill" pathogens: UV-based and chemical-based disinfectants.

UV lamps are classified into three types: UV-A (315 to 400nm), UV-B (280 to 315nm), and UV-C (100 to 280nm). UV-A and UV-B rays cause sunburns in humans. UV-C lamps are extensively used in disinfection as their radiation has a shorter wavelength and consecutively emits a greater amount of energy [63].

Sodium hypochlorite (NaClO), hypochlorous acid (HOCl), chlorine dioxide (ClO2), hydrogen peroxide (H2O2), alcohol, and ozone (O3) are some liquid-based agents used alongside robots to disinfect public spaces [65].

4.1.1 UltraBot

UltraBot (Figure 4.1) is a robot developed by researchers from the Intelligent Space Robotics Laboratory, Space CREI, Skolkovo Institute of Science and Technology in Moscow, Russia. This robot aims to complete indoor disinfection activities using UV-C lamps, decreasing the total bacterial count by 94% after ten minutes of exposure to radiation at a distance of 2.8 meters. This robot can operate autonomously in unknown and known environments. It also has a manual control mode that allows the user to drive the robot by joystick [66].

Figure 4.1: UltraBot Robot [66]

The vehicle has two light detection and ranging (LIDAR) sensors, ten ultrasonic sensors and four cameras. UltraBot uses the ROS package move_base[1] as the main component of its navigation. The robot uses Google Cartographer[2] [67] to perform SLAM when the map of the environment is not known *a priori*. Otherwise, when an environment's model is known, the ROS package `map_server`[3] provides the model to the robot and the ROS package `amcl`[4] is used to perform the robot's localization.

[1] http://wiki.ros.org/move_base
[2] http://wiki.ros.org/cartographer
[3] http://wiki.ros.org/map_server
[4] http://wiki.ros.org/amcl

4.1.2 Smart Cleaner

Like the previous robot, the Smart Cleaner (Figure 4.2) aims to disinfect indoor environments effectively. This robot, developed by researchers from the Department of Electromechanical Engineering in University of Macau (China), uses a hydrogen peroxide atomisation device to eliminate pathogens. A reduction of the total number of air bacteria by 83.7% and airborne fungi by 34.9% was achieved on the tests, which exhibited great results [68].

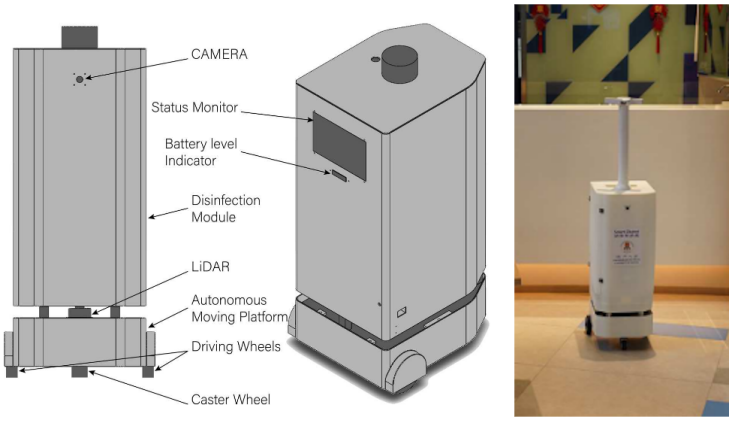

Figure 4.2: Smart Cleaner Robot [68]

Smart Cleaner uses a camera and a LIDAR sensor to navigate autonomously in the environment. The vehicle uses the ROS platform to navigate autonomously. The ROS package `amcl` is used to estimate the robot's location in relation to the environment, together with the ROS package `openslam_gmapping`[5], which is used to build the model of this same environment, allows simultaneous localization and mapping (SLAM).

Smart Cleaner uses as local planner the DWA algorithm and as global planner the A* algorithm. These two planners are connected to the ROS package `move_base` to accomplish its global navigation task.

4.1.3 Intelligent Disinfection Robot

The Intelligent Disinfection Robot (Figure 4.3) was developed by researchers from Huazhong University of Science and Technology and Shangai Taimi Robot Technology Co. in China. This robot integrates artificial intelligence algorithms and robotic technologies for hospital disinfection. It resorts to several mechanisms, such as an ultraviolet lamp, an ultra-dry mist hydrogen peroxide generator and plasma air purification to efficiently cleanse indoor environments [69].

[5] http://wiki.ros.org/openslam_gmapping

Figure 4.3: Intelligent Disinfection Robot [69]

The vehicle can navigate effectively in the environment using visual radar, laser radar, a self-sensing module and ultrasound radar. Intelligent Disinfection Robot utilizes a combination of visual-inertial monocular Simultaneous Localization and Mapping (SLAM) with map reuse, a specific indoor global positioning method, and a technique for fusion of multiple time-of-flight sensors. The visual-inertial monocular SLAM with the map reuse method allows the robot to localize itself and create a map of its environment. This technology uses visual information from a camera and inertial measurements to estimate the robot's motion and create a map of its surroundings. The map reuse feature allows the robot to use previously created maps to improve its localization and mapping performance [70].

The indoor global positioning method used by Intelligent Disinfection Robot enables it to locate itself accurately within indoor environments where GPS signals may be weak or unavailable [71]. In addition, the robot uses a technique to detect surrounding obstacles in real-time by fusing data from multiple sensors, providing a more accurate and reliable detection of obstacles than the direct use of LIDAR or simply fuse ultrasound sensors using artificial rules [72].

4.1.4 UVD Robot

The UVD Robot[6] was a product launched by the company UVD Robots, part of Blue Ocean Robotics, focusing on developing robots to enhance environmental cleaning. This vehicle also uses UV-C technology to perform the disinfection [73].

[6]https://uvd.blue-ocean-robotics.com/features

Figure 4.4: UVD Robot [73]

The robot has laser scanners and 3D cameras that detect impediments along its path. The operator may control the device via tablet using the cameras.

4.1.5 Ava's UV Disinfection Robot

Ava's UV Disinfection Robot[7] is made by Ava Robotics and offers effective, hands-free disinfection. It has intelligent features for planning, execution and reporting. By its name, it is easy to understand that this robot uses UV-C as a method to disinfect the environment [74]. The robot uses LiDAR, 3D cameras, and an inertial measurement unit to achieve fully-autonomous navigation.

Figure 4.5: Ava's UV Disinfection Robot [74]

4.1.6 Comparison between Disinfection Robots

The aforementioned disinfection robots use different algorithms and disinfection methods to achieve the same objective. Having their features compared gives us an overview of their capabilities and methods used, which can help us understand what could possibly be used in this work. The comparison is presented in Table 4.1.

[7] https://www.avarobotics.com/disinfectionrobots

Robot's Name	Localization	Mapping	Navigation	
			Local Planner	Global Planner
UltraBot	AMCL	Google Cartographer	Google Cartographer	Not Available
Smart Cleaner	AMCL	Gmapping	DWA Algorithm	A* Algorithm
Intelligent Disinfection Robot	Indoor global positioning method for service robots	Not Available	Method for fusion of multiple time-of-flight sensors for service robots	Visual-inertial monocular SLAM with map reuse
UVD Robot	Confidential			
Ava's UV Disinfection Robot	Confidential			

Table 4.1: Comparison between robots and their used algorithms. (Some of them keep their software confidential.)

4.2 Coverage Path Planning

The process of establishing a path that traverses over all points of an area or volume of interest while avoiding obstacles is known as Coverage Path Planning (CPP) [75]. Section 2.4 already covered some algorithms developed for this process, but the goal of this section is to present two ROS packages used in CPP. The methods used for achieving full coverage of the environment are different. While the `full_coverage_path_planner` package uses a technique based on [76] to compute the global plan and the `tracking_pid` package to generate the local plan, the `explore_lite` package is dependent on the configuration of the `move_base` node to obtain the global and local plans.

4.2.1 `explore_lite` package

This ROS package[8], developed by Jiří Hörner [77], provides a ROS node for frontier-based exploration. After starting the node, it will continuously run until the robot cannot find more frontiers. The nonexistence of frontiers indicates that the robot covered all the environment. The exploration node on the `explore_lite` package was designed to be a light-weighted version of the node with the same function on the code of the `explore` ROS package [9]. This node correctly handles ROS and `tf` namespaces, allowing several robots under the same ROS Master. The mobile navigation of the robot is made by using the ROS standard stack throw the `move_base` package. The ROS navigation stack uses a local costmap to search for frontiers and find paths to frontiers from the robot's position. The costmap created by `move_base` node is built using the `Costmap2DROS` framework[10] which uses ray-tracing from scans. This approach causes an unnecessary overhead when using those costmaps to search for frontiers. Local costmaps built from a SLAM constructed map produce better results compared to the previous ones mentioned. An alternative to this method is provided in this package. A

[8] http://wiki.ros.org/explore_lite
[9] http://wiki.ros.org/explore
[10] http://wiki.ros.org/costmap_2d

costmap_client node obtains a local costmap subscribing to a map source in the ROS created by a mapping algorithm.

The frontier search algorithm weights all the encountered frontiers and chooses the one with the biggest weight as the next target. This frontier cost is computed through a patch extending `NavfnROS` planner[11] in the core ROS navigation stack.

4.2.2 `full_coverage_path_planner` package

This ROS package contains a Full Coverage Path Planner (FCPP) implementation based on the BSA presented in Section 2.4.2.

`full_coverage_path_planner` provides a `.launch` file that triggers the navigation planning. It starts by generating the coverage path, and after that, "feeds" the local planner with that data in order to trigger the computation of velocity commands necessary to follow the path. Those velocity commands are applied to the wheels to properly maneuver the robot. The `full_coverage_path_planner` package uses two other ROS packages to fulfill its purpose:

move_base_flex (MBF) package $\longrightarrow$ It is a replacement for the `move_base` package, which is the central module of the navigation stack in ROS. MBF offers modular actions for executing plugins for path planning, motion control, and recovery. More about this package in [Putz 2018].

tracking_pid package[12] $\longrightarrow$ It is a tuneable PID control loop to follow a trajectory accurately. It is used as a local planner on 4.2.2.

[11] http://wiki.ros.org/navfn
[12] http://wiki.ros.org/tracking_pid

CHAPTER 5

Disinfection of Spaces

5.1 Introduction

At the beggining of this dissertation, a study on different methods of disinfection using autonomous robots was conducted and presented in Chapter 4.1. Also, a survey of existing coverage and mapping path planners was made and presented in Section 4.2. The objective of this analysis was to give us a general idea of the algorithms used for robot disinfection and to understand what could be used in this work since the development of a coverage algorithm for disinfecting spaces from scratch is a complex project.

The reason for not using the real robot in this work is due to the approach chosen to control the coverage of the disinfected area (explained in Section 5.4.4), which requires the real UV sensor to be installed on top of the robot. I chose Gazebo as the simulation environment not only because there is a lot of documentation to help with development on this platform, but also because there is a GitHub repository with a ROS package that allows the use of ROS alongside the MiR robots and already has launch files to use them in Gazebo Simulator.

5.2 MiR100 Robot's Overview

The MiR100 is a mobile robot manufactured by Mobile Industrial Robots (MiR), a Danish company specialist in producing autonomous mobile robots for industrial and commercial use. The robot is compact, highly maneuverable, and can support payloads of up to 100 kg, making it suitable for the installation of an ultraviolet light's emitter. The vehicle is equipped with laser scanners, 3D cameras, ultrasonic sensors, infrared sensors and wheel encoders, which it enables the robot to navigate autonomously and avoid obstacles in its path. MiR100 is powered by an onboard computer that runs on ROS allowing its integration with a range of different systems and technologies. A repository that provides ROS packages that simulate the robot and its behavior is already available online, and it will be used in this thesis.

5.2.1 The `mir_robot` repository

A GitHub repository called `mir_robot`[1], created by the German Research Center for Artificial Intelligence, contains a ROS driver and ROS configuration files for the MiR Robots. This repository is composed of six ROS packages. In `mir_actions` package, there are the ROS actions definitions for the robot. The `mir_description` package, where the URDF files for the robot's model are used, not only, in Rviz simulated environment for the graphic representation, but also, to define the position of the several parts of the robot essential for many tasks. The `mir_dwb_critics` package contains plugins for the `dwb_local_planner` ROS package which is a dynamic window local planner used in the Gazebo simulator. The `mir_driver` package, which is a reverse ROS bridge to connect to the MiR Robot. The `mir_gazebo` package, which has the configuration files and the simulation launch for the robot. The `mir_msgs` package, which has the ROS message definitions, and finally, the `mir_navigation` package, where are the configuration and launch files of the ROS package `move_base`.

5.3 Overall Architecture of the Project

Figure 5.1: Illustration of the Overall Architecture of the Project

In order to build a disinfection system for public spaces, we turned to existing GitHub repositories, represented by the Grey rectangles in Figure 5.1, where ROS packages with

[1]https://github.com/DFKI-NI/mir_robot

useful features for this project are accessible. The `mir_robot` repository, explained in Section 5.2.1, not only makes it possible to use ROS alongside the MiR100 robot, but also provides physical and mechanical's virtual models of the robot itself together with launch files of a simulated Gazebo environment. The `full_coverage_path_planner` repository, already introduced in Section 4.2.2, makes it possible to obtain the path that the robot has to travel in order to cover a certain map known a priori. The MiR100's radius and the radius of the UV sensor ray range are the input parameters used to obtain the coordinates of the path. The `iris_lama_ros` repository is nothing more than the integration with ROS of the `iris_lama` repository's features explained in Section 2.6. In this repository, the Scan Matching-based localization method, which is best explained in Section 2.1 , is the one used in this work. A fifth repository called `mir100_germIrrad` was also used, but unlike the previous ones, this was created from scratch for project's purposes and contains ROS packages where specific functionalities are implemented.

Analyzing Figure 5.1 on a ROS package level, we have red, green, blue and purple rectangles. The red rectangles represent those that are not used in this project. In green are the packages that are used and have suffered project related changes. In blue are the packages that are used in this work and have not been modified. Finally, in purple are the ROS packages created from scratch. A rectangle as a dashed contour means that the ROS package is external to the GitHub repository.

This section was more directed towards demonstrating the design of the project, while Section 5.4 will focus on the operation of the system at the ROS computation graph level (Section 3).

5.4 System Architecture

Figure 5.2 describes at a deeper level how the developed system works. Through a ROS node called `map_server` it is possible to provide the map of the environment we want to disinfect. This ROS node publishes the data in the `/map` topic to which the `move_base_flex` node and `irradiation_map` node, that will explained in 5.4.1, subscribe. The first step is to estimate the pose of the robot and to do that the `loc2d_ros` node that will be explained in Section 5.4.2. Then we can start the navigation plan described in Section 5.4.3. The `gazebo` node also publishes the odometry of the simulated MiR100 on `/odom` topic, which is subscribed to by the `move_base_flex`. Although the `move_base_flex` node is not part of the ROS navigation stack, unlike the `move_base` node, its functionality is to replace it while being compatible with plugins developed for `move_base`. The Spiral-STC plugin is used by `move_base_flex` as a planner. An explanation of this method is given in Section 5.4.3.

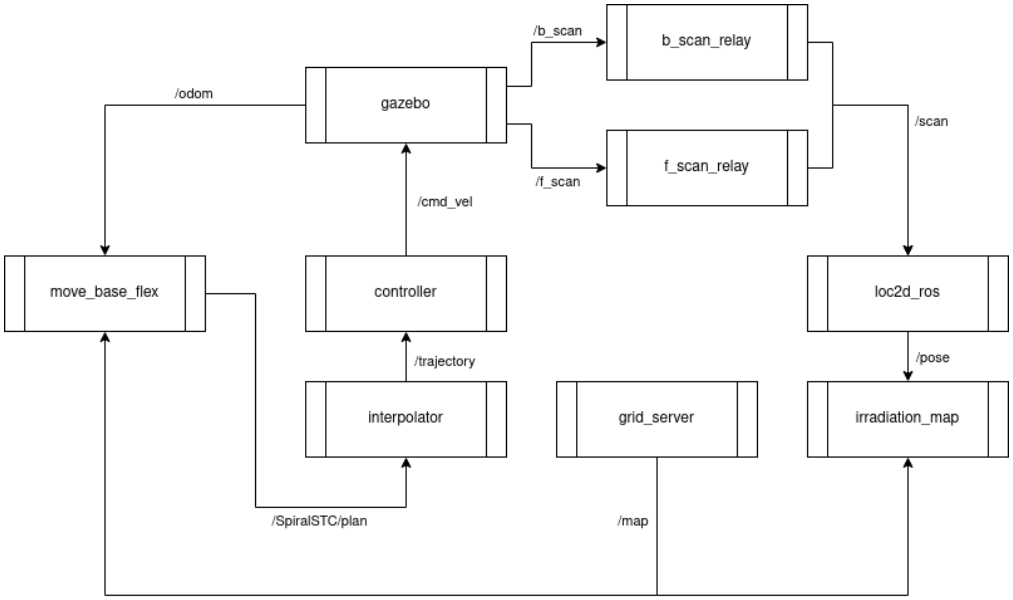

Figure 5.2: System Architecture Diagram at ROS Computation Level

5.4.1 Sensorization

SICK S300 Laser Scanners

This project used two SICK S300 LiDARS laser sensors, one mounted on the front left of the robot and the other on the rear right. Their position is only capable of detecting objects that intersect a plane 20cm above the floor. It is important to note that, since LiDAR sensors work with flight times, there are situations that can influence the validity of the data they collect. An obstacle being transparent or having reflective properties is one such situation; another is exposing the sensor to strong direct light, thus causing it to detect non-existent obstacles.

Figure 5.3 shows the coverage area of the two sensors on the real MiR100. We can see that each LiDAR has a 270º field of view, overlapping on the front right and rear left, thereby providing a 360º coverage around the robot.

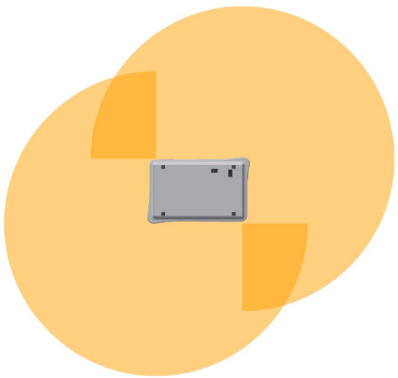

Figure 5.3: Field of view of the LiDAR sensors on the real MiR100 [78]

The sensors on the virtual MiR100 simulate the same behavior as those on the real robot, offering 360º coverage, as can be seen in figure 5.4. The image on the left proves the aforementioned, which results from the junction of the areas covered by the left front sensor (middle image) and the right rear sensor (right image).

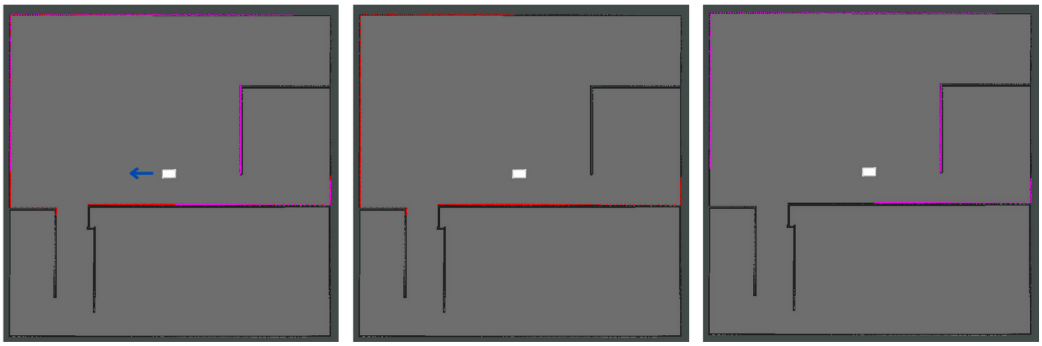

Figure 5.4: Field of view of the LiDAR sensors on the simulated MiR100 (left); Field of view of the front left sensor (middle); Field of view of the rear right sensor (right).

UV-Light Irradiation Field

The goal of this component is to represent the area covered by the ultraviolet sensor. This Sensor does not have neither a simulated model nor its own physics since during the development of this work there was no final decision about which ultraviolet sensor would be used.

On the ROS package created, the estimated value of the robot pose relative to the map is used to calculate an area that corresponds to the area that is being irradiated by the UV Sensor when the vehicle is in the estimated position with an estimated orientation, as can be visualized in figure 5.5.

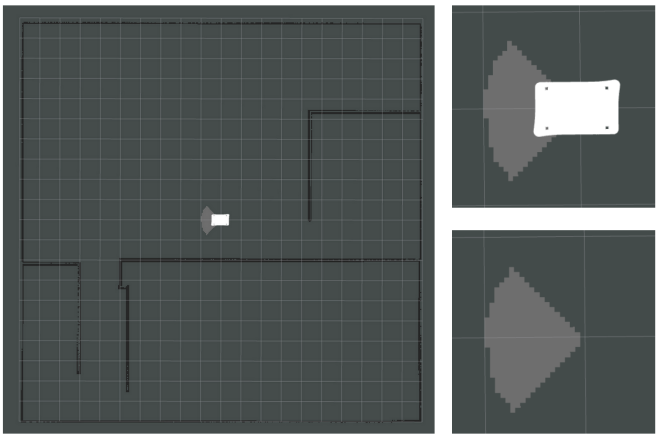

Figure 5.5: Field of view of the LiDAR sensors on the simulated MiR100

In the code, there are four parameters that must be set manually for the program to work correctly. These parameters are the `openAngle`, the `dirAngle`, the `rangeUV` and the `poseSensor`.

- `openAngle` refers to the angular extent of ultraviolet radiation emitted by the sensor;
- `dirAngle` refers to the orientation in which the sensor is pointing;
- `rangeUV` refers to the maximum distance reached by the ultraviolet radiation;
- `poseSensor` refers to the location of the sensor relative to the robot.

Aperture of the UV Sensor

The aperture of the UV Sensor was set to 90º in order to optimize the emission of radiation so the intensity of radiation could be concentrated in a narrow field.

Orientation of the UV Sensor

The orientation of the UV sensor is configured using the degrees relative to the orientation of your robot. This means that we need to specify the angle at which the sensor is positioned in relation to the robot's direction. For instance, 0 degrees denotes the robot's forward direction, 90 degrees represents the right side, 180 degrees represents the backward direction, and 270 degrees represents the left side. If the UV sensor is mounted at a 45-degree angle relative to the robot's forward direction, it means the sensor is positioned diagonally between the forward and right directions. Figure 5.6, taking into consideration that the robot is directed to the right, the left image shows an example of a `dirAngle` set to 180 degrees and the right image shows a `dirAngle` of 90 degrees. This parameter's configuration will be justified in Section 6.2.

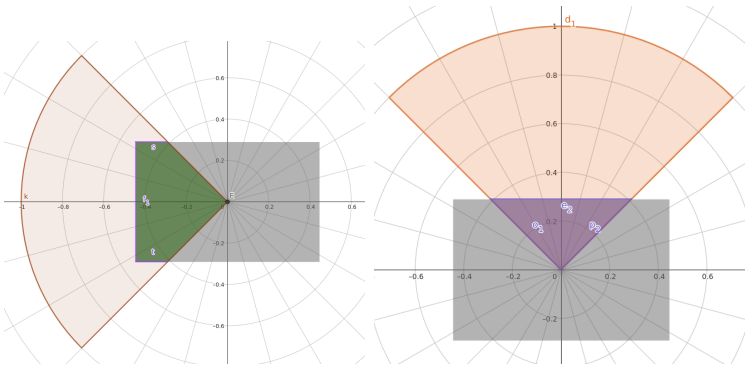

Figure 5.6: Example of a orientation of UV Sensor of 180º (left image), and a orientation of 90º (right image).

Range of the UV Sensor

The range of the UV sensor on the robot is vital for ensuring that all surfaces and areas are adequately exposed to germicidal UV light. A sensor with a longer range can detect and disinfect surfaces that are farther away, enhancing coverage. This parameter's configuration will be justified in Section 6.4.

Sensor Position relative to the Robot Position

The UV Sensor is installed near the center of the top of the robot, slightly shifted to the rear. Considering that the UV Sensor weight around 50kg. This specific position of the sensor was chosen based on the MiR100 robot user manual (Figure 5.7), to make sure that both the Sensor and the robot are stable during operation. The MiR100 robot user manual was also consulted to ensure that the Sensor position was compatible with the manufacturer's specifications and did not interfere with other robot functionality.

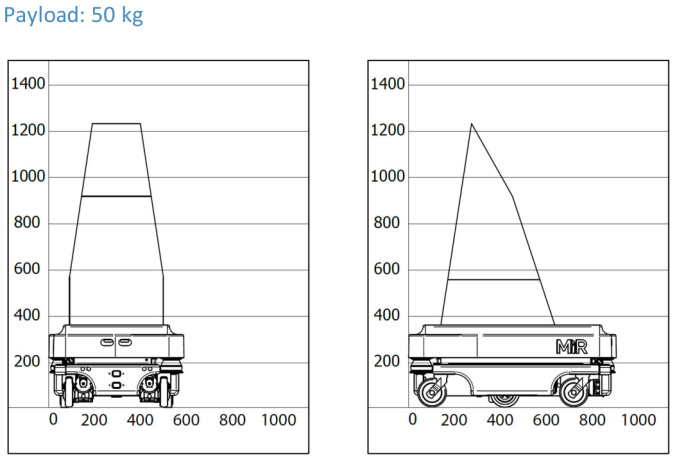

Figure 5.7: Position of a payload of 50kg on the MiR100 Robot [78]

5.4.2 Localization

In this project, the "Efficient Localization Based on Scan Matching with a Continuous Likelihood Field" [11] is used as the localization method. This technique was developed at IRIS Lab and it uses scan-matching (explained in section 2.1.2).

A key component of this implementation was the usage of the ROS node `loc2d_ros`, which served as the core of our localization system. This node, available on the `iris_lama_ros` repository, housed the algorithms necessary for accurately estimating the robot's pose.

To integrate LiDAR Sensors (section 5.4.1) data, the `loc2d_ros` node subscribed to the ROS topic `/scan`, where fused data from the two sensors mounted on the robot was published. This data integration was crucial, providing a comprehensive understanding of the robot's surroundings. The fused sensor data served as input for the localization method.

Once the pose estimation is successfully completed, the `loc2d_ros` node publishes it on the ROS topic `/pose`. This timely publication allowed other components of our robotic system to access accurate localization information, empowering them to make informed decisions based on the robot's position and orientation.

5.4.3 Navigation

In this project, for navigation purposes, we use the `move_base_flex` ROS package to give the necessary movement orders to the robot in order to reach the target point on the map. This versatile package provides a robust framework for autonomous robot navigation, being even more modular and customizable compared to the original `move_base` package. It allows users to configure and replace components like planners, controllers, and costmaps easily. The combination of these components is responsible for generating the navigation path that is applied to the robot in order to cover the maximum possible area of the map.

In this work, we opted to utilize a coverage planner instead of a global planner. The reasoning behind this choice was rooted in the primary objective of our robotic system: to

comprehensively cover the entire area of the map for disinfection purposes rather than simply reaching a specific goal point via the least costly route.

Planner - Coverage Planner

A crucial element of our navigation strategy was the choice of a Global Planner, an important decision that defines the robot's high-level path planning. In this project, we opted for the backtracking spiral algorithm (Section 2.4.2). This component is implemented by using a `.launch` file from the `full_coverage_path_planner` GitHub repository (Section 4.2.2), designed to trigger the coverage planner. Within the `.launch` file, several input arguments were incorporated, each tailored to optimize performance and precision in mapping and navigation. They are `map`, `coverage_area_offset`, `coverage_area_size_x`, `coverage_area_size_y`, `target_x_vel`, `target_yaw_vel`, `robot_radius`, `tool_radius` and `rviz`.

The `map` argument references a path to a `.yaml` file containing a representation of the static objects of the environment, providing insights into its layout. Additionally, the parameters `coverage_area_size_x`, and `coverage_area_size_y` were both configured as 20m. These parameters define the dimensions of the coverage area within the map. The `target_x_vel` and `target_yaw_vel` parameters define the robot's velocities during forward movement and directional changes, respectively. Fine-tuning these values is essential to achieving a comprehensive correlation between the duration and quality of disinfection. These parameters configuration will be justified in Section 6.3.

The `robot_radius` parameter is used to maintain safe distances from static obstacles. According to the MiR100 website[2], the operational corridor width is 1m and the traversable gap tolerance is 0.02m, which together make a minimum traversable gap width of 1.02m. This parameter was configured as 0.53114m because of the configuration example of figure 5.9, and since this value is not less then half of the minimum traversable gap width, its use is totally acceptable. Figure 5.8 describes the difference between the value chosen and the minimum traversable gap width.

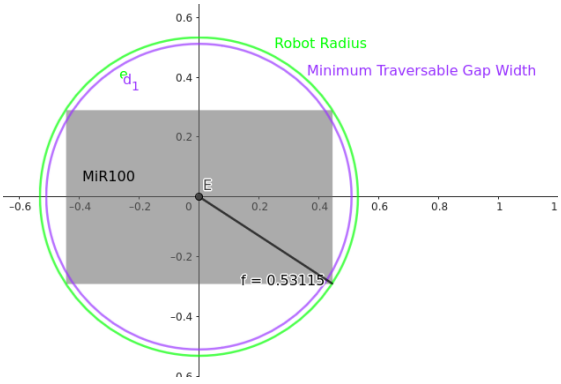

Figure 5.8: Representation of the minimum traversable gap with and `robot_radius` parameter values.

Furthermore, the `tool_radius` parameter was used in the discretization of the space

[2]https://www.mobile-industrial-robots.com/solutions/robots/mir100/

so the algorithm could plan a comprehensive coverage strategy, accounting for the robot's physical dimensions. The value of this parameter needs to be less or equal to `robot_radius` and higher than 0.29m as per the configuration example in figure 5.9. Figure 5.10, on the left, illustrates the path calculated for a `tool_radius` of 0.53115m and on the right for a `tool_radius` of 0.29m using the same radiation range (Section 5.4.1). It can be seen that increasing the parameter value causes a decrease in the area that is irradiated more than once. This parameter's configuration will be justified in Section 6.2.

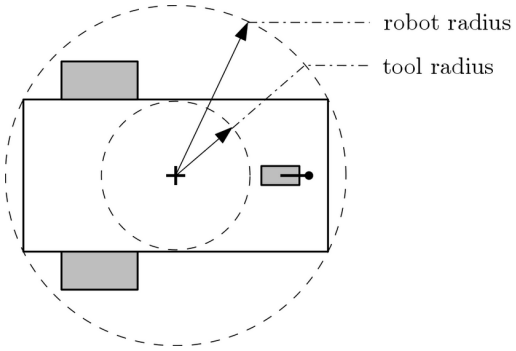

Figure 5.9: Illustration of possible configuration for `robot_radius` and `tool_radius` parameters [79].

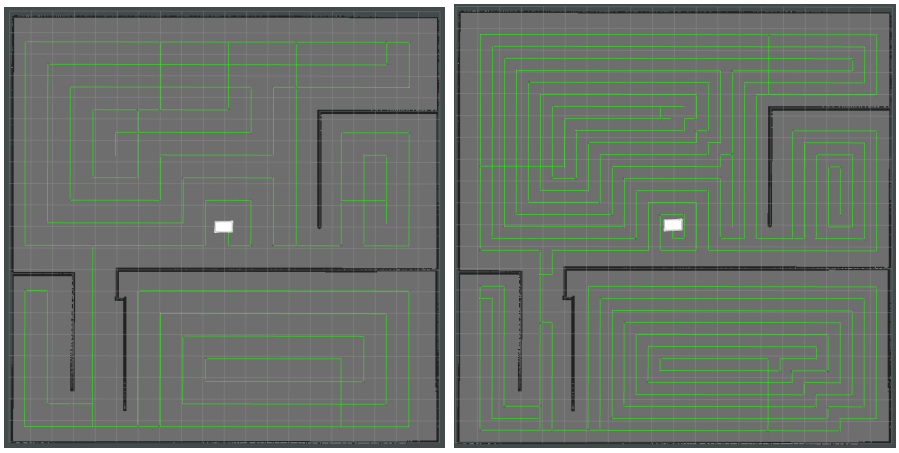

Figure 5.10: Example of a coverage path generated using a `tool_radius` of 0.53115m (left image) and a `tool_radius` of 0.29m (right image).

Additionally, the `rviz` parameter allows us to enable or disable the launch of the RViz tool (Section 3.1.1) with a project specific configuration providing real-time visual feedback on the robot's movement, sensor data, and coverage progress.

In our project, we focused on optimizing specific parameters, including `target_x_vel`, `target_yaw_vel` and `tool_radius`. This parameter tuning process was crucial. It ensured that our robot navigated efficiently, comprehensively covering the designated area.

Controller - Local Planner

A specialized local planner known as `tracking_pid` was used in the context of this work.

This local planner, on the other hand, has a limitation: it cannot avoid non-stationary objects. Unlike classic local planners (Section 2.5), which are good at avoiding obstacles, this planner works more like a controller. Its operation can be described using the following mechanisms: Tracking Proportional-Integral-Derivative (PID)'s core functionality is its ability to implement a tunable PID control loop, ensuring accurate adherence to a predefined trajectory. This is accomplished with an interpolator, which moves a goal along a specified path at a customizable velocity.

In one of its tracking modes, the planner deploys a concept known as the "carrot", which is positioned at a specific distance in front of the robot. This carrot serves as a reference point for computing velocity commands, taking into account both lateral and longitudinal errors between the current Global Point (GP) and the Control Point (CP) on the trajectory. By analyzing these errors, the planner adjusts the robot's movements to maintain accurate trajectory tracking. Figure 5.11 illustrates this mode.

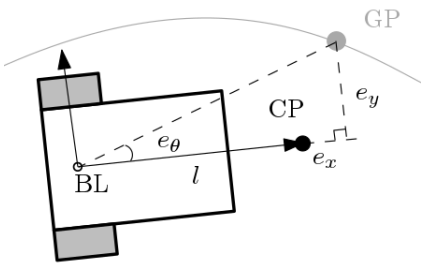

Figure 5.11: Representation of the "carrot" strategy of `tracking_pid` [80].

On the second tracking mode, if a smooth path is provided, the controller offers the flexibility to directly track the path using the robot's `base_link`, eliminating the need for trailing a carrot point. In this scenario, a Projected Global Point (PGP) is computed, serving as the target for the Control Point (CP) to track. Additionally, the planner can incorporate yaw error as a control input, further enhancing its ability to precisely follow the specified path. Figure 5.12 illustrates this mode.

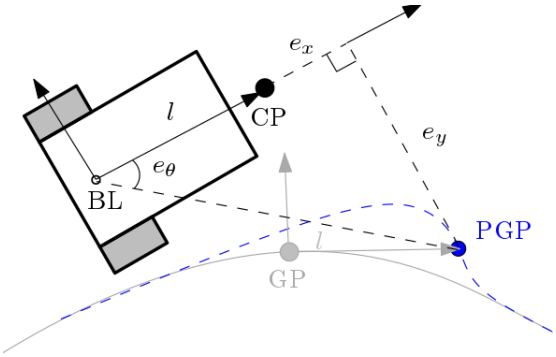

Figure 5.12: Representation of the projected global point (PGP) strategy of `tracking_pid` [80].

In essence, this local planner, while lacking obstacle avoidance capabilities for dynamic

obstacles, excels as a robust controller. Its utilization of PID control loops, coupled with versatile tracking options such as carrot-based tracking and direct path tracking with yaw error consideration, underscores its effectiveness in ensuring accurate and efficient trajectory following within the specified system.

The `tracking_pid` ROS package introduces two nodes: the interpolator and controller (Figure 5.2). The `interpolator` node assumes a central role in the trajectory following process. Subscribing to the `/SpiralSTC/Plan` topic, this node receives the calculated path from the coverage planner. At a given velocity, this node moves a goal over the acquired path and calculates the control point for that exactly goal.The control point that results from this process is then published under the `/trajectory` topic.

Complementing the `interpolator`, the `controller` node uses the trajectory data by subscribing to the `/trajectory` topic, and a PID controller to calculate the velocity commands necessary to stay on track. These commands are then published under the `/cmd_vel` topic. Finally, the `/cmd_vel` topic is subscribed to by the `gazebo` node where the calculated velocity commands are used by a simulated robot operating within the Gazebo framework to synchronize its movements so it traverse the coverage path.

5.4.4 Disinfectioned Area Coverage

One of the key challenges in this domain lies in ensuring comprehensive coverage of the target area by the UV Sensor, guaranteeing that every spot is disinfected. To tackle this challenge, an approach has been employed, involving the generation and utilization of a `SimpleOccupancyMap` from the `iris_lama` repository (section 2.6).

At the heart of this strategy is the concept of an `Occupancy Map` or `Occupancy Grid` (explained in section 2.2.1), a grid-based representation of an environment where each cell in the grid represents the occupancy status of a corresponding area in the physical world. In this case, the area requiring disinfection is mapped out, and the free cells within this map signify the regions that have already been covered by the robot.

The `SimpleOccupancyMap` is a dynamic data structure that is continuously updated. Every time the robot changes its pose, the occupancy map is recalculated to reflect the altered landscape. This constant update is crucial to maintaining an accurate representation of the covered area. At the ROS level, the process begins with the ROS node `irradiation_map`, which subscribes to the topic `/pose`. This topic serves as an abstract channel through which the estimation of the robot's pose is transmitted. The robot's pose data is essential in determining its position and orientation within the environment, enabling the accurate updating of the current UV Sensor irradiated area (section 5.4.1) and respectively the occupancy map.

Upon receiving the updated pose information, the ROS node processes this data and subsequently publishes the updated `SimpleOccupancyMap` onto the topic `/map_irradiated`. This topic acts as a broadcast channel, making the occupancy map data accessible to other components of the robotic system or any external entities requiring this information. By making the occupancy map publicly available, other modules or systems can leverage this data for various purposes, such as visualization, analysis, or decision-making processes.

5.5 Summary

In this Chapter we gave a brief summary about the characteristics of the real MiR100 and covering the functionalities offered by the `mir_robot` GitHub repository. We present the developed system for indoor robot disinfection using the simulated MiR100. We provided a more detailed description of each ROS package used as well as which parameters can be tuned in order to improve our disinfection. Furthermore, we discussed the navigation plan and how the current disinfected area coverage will be controlled during operation. Finally, we present a functional system that is able to disinfect a static space without dynamic obstacles.

CHAPTER 6

Results and Troubleshooting

6.1 Methodology of the tests

The goal of this section is to explain which test scenarios and selection criteria were used to evaluate the efficiency of disinfection and tune the system to achieve the best possible disinfection. All the tests in this section are done in a simulated environment. Figure 6.1 presents the map used in the tests.

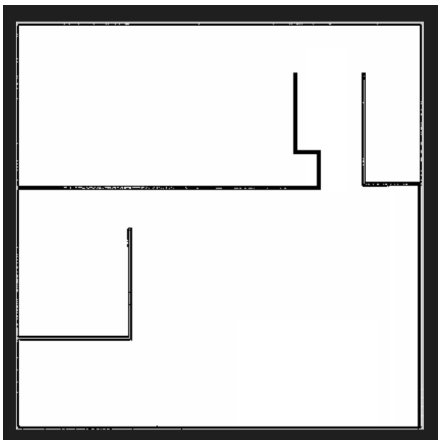

Figure 6.1: Test scenario used.

The behavior of the disinfection will be recorded and evaluated according to the following metrics:

- **Disinfected Area:** Percentage of area that was irradiated by the UV Sensor.
- **Disinfection Time:** Time it takes for the MiR100 to cover the environment.
- **Overlapping Irradiated Area:** Area that is irradiated more than one time.
- **Irradiation Exposure Time:** Time of irradiation exposure per square meter of area.

Several parameters play a crucial role in determining the effectiveness and efficiency of a disinfection robot. These parameters directly impact the coverage of disinfection and the

Parameter	Value Range
`target_x_vel`	[0 , 1.5] (m/s)
`target_yaw_vel`	[0 , 1.5] (m/s)
`tool_radius`	[0.29 , 0.53115] (m)
`rangeUV`	[1 , 2] (m)
`dirAngle`	[0 , 360[(deg)

Table 6.1: Value range of the parameters manipulated during tests.

time required to complete the task. Table 6.1 lists the parameters that will be manipulated in this test phase alongside their value range. Optimizing these parameters involves striking a balance between coverage, speed and and time exposure. A well-calibrated robotic system, tailored to the specific environment it operates in, ensures effective disinfection coverage while minimizing the time required to complete the task without the minimum amount of time exposure necessary to kill pathogens.

6.1.1 Method of Evaluation of the Disinfection

In order to obtain the metrics necessary to evaluate the developed system the `specs` ROS node was created. Figure 6.2 illustrates at the ROS communication level the implementation. This node is connected to the system presented in Section 5.4.

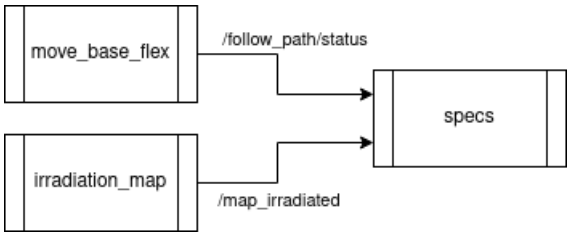

Figure 6.2: Testing Architecture Diagram at ROS Communication Level.

The following procedure was used to calculate the percentage of disinfected area. The node created subscribes the `/map_irradiated` topic and gets the occupancy grid published by the node `irradiation_map` there. With that data, we can obtain a $TotalCells$ of 157609 cells and a $OccupiedCells$ of 5044 cells, knowing that these two values will remain immutable as long as the map used is not modified. The $FreeCells$ value is recalculated, using Listing 6.1, every time a new occupancy map is published in `/map_irradiated`. Equation 6.1 presents the final calculation to obtain the percentage of disinfected area.

```
void coverageStatusCallback(const nav_msgs::OccupancyGrid& msg)
{
    int numCellsFree = 0;

    for (size_t i = 0; i < msg.data.size(); i++)
    {
        if(msg.data[i] == 0)
        {
```

```
            numCellsFree++;
        }
    }

    if (numCellsFree > Coverage)
    {
        Coverage = numCellsFree;
    }
}
```

Listing 6.1: Callback function for area irradiated occupancy grip update.

$$DisArea = \frac{FreeCells}{(TotalCells - OccupiedCells)} \times 100 \tag{6.1}$$

The disinfection time was obtained using the implementation in Listing 6.2. The created node subscribes to the `/follow_path/status` topic and gets the *StartTime* when the goal starts being processed by the action server, and *EndTime* when it reaches the end of the planned path. Equation 6.2 presents the calculation for value of disinfection time in the format of minutes:seconds.

```
bool startConditionMet = false;
void goalStatusCallback(const actionlib_msgs::GoalStatusArray::ConstPtr& msg)
{
    for (const auto& status : msg->status_list)
    {
        if (status.status == 1)
        {
            if (!startConditionMet)
            {
                startTime = ros::Time::now();
                startConditionMet = true;
            }
            break;
        }
        else
        {
            endTime = ros::Time::now();
            startConditionMet = false;
            break;
        }
    }
}
```

Listing 6.2: Callback function for goal status update.

$$DisDuration = StartTime - EndTime \tag{6.2}$$

The overlapping irradiated area metric was evaluated because, for a specific area and a constant intensity of UV radiation, irradiating the same area for more than one time will be more efficient. This metric was analyzed from a qualitative point of view because of the type of occupancy map used (Section 5.4.4) does not allow us give a different classification to those cells. We know that the `tool_radius` and `rangeUV` parameters are the ones that may have more influence in this classification and Figure 6.3 helps us understand why. In case of `tool_radius`, it defines how close from the parallel path the robot will pass (Figure 6.3 top images), and in case of `rangeUV`, it defines if a the UV ray will reach an bigger portion of area previously irradiated or not (Figure 6.3 bottom images).

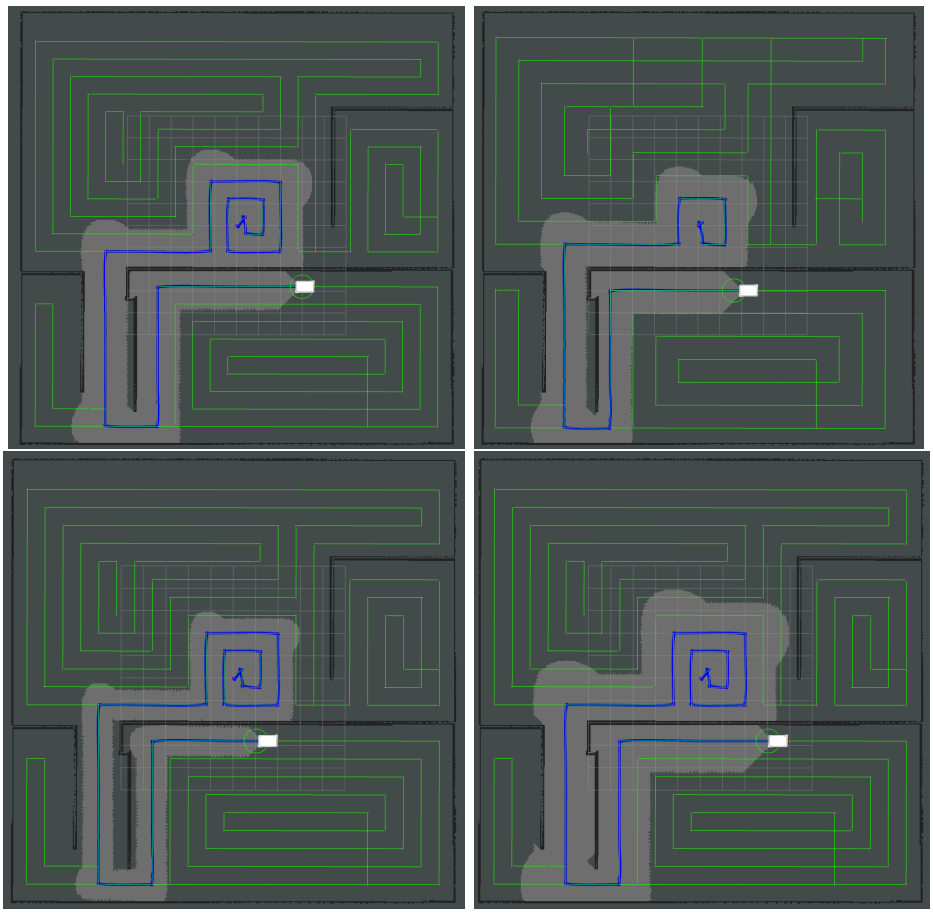

Figure 6.3: Example of situations where overlapping irradiated area can vary because of different `tool_radius` (Top Left=0.4106m; Top Right=0.53115m) or different `rangeUV` (Bottom Left=0.5m; Bottom Right=2m)

The irradiation exposure time metric was also evaluated in a qualitative way. This metric is used because, for a specific area and a constant intensity of UV radiation, a longer exposure will inactivate more pathogens. Here, the parameters that have more influence are `target_x_vel` and `target_yaw_vel` due to being the ones that control the velocity of the robot.

6.2 Evaluation of the UV Sensor's orientation and `tool_radius` parameter

In this first phase of tests, we determined the most optimal orientation for the UV Sensor to be employed in tandem with the best-suited value for the parameter `tool_radius` used for the path planner (Section 5.4.3). The goal of this section is to understand how parameters `tool_radius` and `dirAngle` influence the percentage of disinfected area and disinfection time.

The approach chosen was to divide by 4 the set of values for `tool_radius` in Table 6.1 obtaining 0.29m, 0.3503m, 0.4106m, 0.4709m and 0.53115m as the values to be tested. Regarding the parameter `dirAngle` that represents the UV Sensor's orientation (Section 5.4.1), tests were performed for a degree of 0, 90, 180 and 270. In order to achieve the goal, we assigned to the rest of the parameters mentioned in Section 6.1.1 the constant value presented.

- `rangeUV` = 1.5m;
- `target_x_vel` = 1m/s;
- `target_yaw_vel` = 1m/s;

Table 6.2 presents the results from the tests made in the simulator.

		Disinfected Area (%)				**Disinfection Time (min:sec)**			
		dirAngle				**dirAngle**			
		90º	**0º**	**270º**	**180º**	**90º**	**0º**	**270º**	**180º**
tool_radius	**0.29**	96.79	97.30	84.68	99.30	36:59	36:59	37:01	37:01
	0.3503	94.78	95.99	81.74	98.69	28:34	28:34	28:35	28:35
	0.4106	92.94	96.62	81.52	97.12	22:14	22:15	22:16	22:16
	0.4709	64.04	68.50	56.59	69.52	15:55	15:54	15:56	15:56
	0.53115	90.28	98.14	80.91	97.23	19:01	19:01	19:03	19:02

Table 6.2: Percentage of disinfected area and the corresponding disinfection time for different values of `tool_radius` and `dirAngle`.

By analyzing the data, we can confirm that the disinfection time is independent from the UV Sensor's orientation. This statement took us to focus mainly on the percentage of disinfected area. The green values represent the three best disinfected area percentages. The combination of `tool_radius` and `dirAngle` that obtains 99.30% and 98.69% of disinfected area has the advantage of having a low value of `tool_radius` which increases the UV Sensor irradiated area overlapped. On the other hand, their disinfection time is approximately 18 min higher than the fastest time. For the combination that gets 98.14%, its percentage of disinfection area is less than 1.5% lower than the first two and it has the best disinfection time of all. The disadvantage of this combination falls into the high value of `tool_radius` which decreases the UV Sensor irradiated area overlapped. The yellow values represent possible candidates that are not part of the top three but present characteristics that make them possible choices. There are three reasons why we considered them. Firstly, they only take 3 minutes more of disinfection time than the fastest time. Secondly, their percentage of disinfection area is, in worst case, less than 3% of the best coverage. Thirdly, their value of `tool_radius` can still offer a reasonable irradiated area overlapped. When the `tool_radius`

value is set to 0.4709m the planner cannot generate a path that covers the entire map, with the row represented in red.

We decided that the combinations `tool_radius/dirAngle` with values 0.29m/180º, 0.53115m/0º and 0.4106m/180º will be taken to the next section of tests (Section 6.3) where it will be checked how much disinfection time could be gained by manipulating the velocities (Section 5.4.3).

6.3 Evaluation of the MiR100's velocity during forward movement and directional changes

This second phase of tests has the objective of understanding how the variation of the parameters `target_x_vel` and `target_yaw_vel` (Section 5.4.3) corresponding to the forward movement velocity and directional change movement velocity , respectively, affect the disinfection area covered and the disinfection time.

In this section, the idea was to divide by two the range of values for both velocities in Table 6.1 obtaining 0.5m/s, 1m/s and 1.5m/s as values of `target_x_vel` and `target_yaw_vel` to be tested. For the `tool_radius` and `dirAngle` parameters, the candidate `tool_radius/dirAngle` combinations taken from Section 6.2 were used. The `rangeUV` parameter was kept in 1.5m.

Always taking into account a balance between the three criteria evaluated (Section 6.1), the goal was to see if the disinfection time could be improved in comparison with the values in Table 6.2 without affecting the percentage of disinfected area.

Tables 6.3, 6.4 and 6.5 present the disinfection time and percentage of disinfected area results from the tests made in the simulator.

		target_x_vel (m/s)		
		0.5	**1**	**1.5**
target_yaw_vel (m/s)	**0.5**	99.30% 42min:55sec	99.30% 37min:09sec	99.30% 36min:39sec
	1	99.30% 42min:46sec	99.30% 37min:01sec	99.30% 36min:32sec
	1.5	99.30% 42min:44sec	99.30% 36min:59sec	99.30% 36min:31sec

Table 6.3: Results of tests for different values of `target_x_vel` and `target_yaw_vel` and candidate `tool_radius/dirAngle` combination of 0.29m/180º.

		target_x_vel (m/s)		
		0.5	**1**	**1.5**
target_yaw_vel (m/s)	**0.5**	97.40% 22min:17sec	97.39% 19min:06sec	97.36% 18min:51sec
	1	97.39% 22min:11sec	97.39% 19min:01sec	97.37% 18min:50min
	1.5	97.40% 22min:10sec	97.37% 19min:01sec	97.37% 18min:44sec

Table 6.4: Results of tests for different values of `target_x_vel` and `target_yaw_vel` and candidate `tool_radius`/`dirAngle` combination of 0.53115m/0º.

		target_x_vel (m/s)		
		0.5	**1**	**1.5**
target_yaw_vel (m/s)	**0.5**	97.12% 26min:13sec	97.10% 22min:19sec	97.10% 21min:59sec
	1	97.12% 26min:11sec	97.12% 22min:16sec	97.09% 21min:55sec
	1.5	97.10% 26min:10sec	97.11% 22min:15sec	97.10% 21min:55sec

Table 6.5: Results of tests for different values of `target_x_vel` and `target_yaw_vel` and candidate `tool_radius`/`dirAngle` combination of 0.4106m/180º.

Analyzing the collected data, we can obtain three main conclusions. Firstly, the percentage of disinfected area is independent of the velocity during forward movement and directional changes. Secondly, `target_yaw_vel` does not have a significant impact on the disinfection time. Thirdly, disinfection time for `target_yaw_vel` values of 1m/s and 1.5m/s differs by less than one minute in all the tables.

The green cells in Table 6.3, Table 6.4 and Table 6.5 represent the best candidate `target_yaw_vel`/`target_x_vel` combination for each candidate `tool_radius`/`dirAngle` combination obtained from Section 6.2. With that said, the candidate `tool_radius`/`dirAngle` combination of Table 6.3 is discarded because, despite having the best irradiated area overlapping of all three, its fastest disinfection time takes eighteen minutes longer than the overall fastest disinfection time (Table 6.4) and the difference between its percentage of disinfected area (99.30%) and the overall worst percentage of disinfected area (97.09% in Table 6.5) is less than 2.5%. The yellow cells represent candidates that offer approximately the same disinfection time and percentage of disinfected area but are better than the green cells on irradiation exposure time so we decided that candidates `target_yaw_vel`/`target_x_vel` combination from Table 6.4 and Table 6.5 with their result marked in yellow should be the ones passing to the final tests. Now we have to evaluate which candidates should pass to the final phase of tests

6.4 Evaluation of the UV Sensor's Radiation Range

This final phase of test has as its goal to identify how parameter `rangeUV` (Section 5.4.1) influences the disinfection. We divided by two the set of values for the range of the UV sensor in Table 6.1 obtaining 1m, 1.5m and 2m as values to be tested. The parameters' values from the previous section were used here, and the results obtained are shown in the Tables 6.6 and 6.7. With that said, we tested two sets with the following values for the parameters:

Set 1:	Set 2:
`tool_radius` = 0.4106m	`tool_radius` = 0.53115m
`dirAngle` = 180º	`dirAngle` = 0º
`target_x_vel` = 1m/s	`target_x_vel` = 1m/s
`target_yaw_vel` = 0.5m/s	`target_yaw_vel` = 0.5m/s

	`rangeUV` (m)		
	1	**1.5**	**2**
Disinfected Area (%)	92.41	97.10	99.30
Disinfection Time (min:sec)	22:19	22:19	22:19

Table 6.6: Results of tests for different values of `rangeUV` using the configuration of Set 1.

	`rangeUV` (m)		
	1	**1.5**	**2**
Disinfected Area (%)	89.21	97.39	99.25
Disinfection Time (min:sec)	19:04	19:06	19.06

Table 6.7: Results of tests for different values of `rangeUV` using the configuration of Set 2.

The analysis of the tables yields several conclusions. Firstly, the study reveals that the `rangeUV` parameter does not exert any discernible influence on the disinfection time. Secondly, a closer inspection of the data demonstrates a marginal disparity in the percentage of disinfected area when comparing `rangeUV` values of 1.5 meters and 2 meters; the variation is less than 2.5%. However, when contrasted with a `rangeUV` of 1 meter, a notable distinction emerges. In the best-case scenario (Table 6.6), the percentage of disinfected area differs by more than 4.5%, signifying a bigger impact on disinfection outcomes.

Taking into consideration a fundamental principle: the intensity of UV radiation diminishes proportionally as the distance from the emission point increases. We can affirm that a shorter `rangeUV` value correlates with a higher intensity, highlighting a necessary trade-off: while a shorter range allows for more intense UV exposure, it inherently limits the coverage area. Therefore, the range of the UV Sensor chosen was 1.5 meters, and it is marked in green in the Tables 6.6 and 6.7. The two final candidates for parameter configurations are the following:

Candidate 1	**Candidate 2**
`tool_radius` = 0.4106m	`tool_radius` = 0.53115m
`dirAngle` = 180º	`dirAngle` = 0º
`target_x_vel` = 1m/s	`target_x_vel` = 1m/s
`target_yaw_vel` = 0.5m/s	`target_yaw_vel` = 0.5m/s
`rangeUV` = 1.5m	`rangeUV` = 1.5m

6.5 Evaluation of Disinfection

Table 6.8 enumerates the advantages of both final candidates from Section 6.4.

Candidate 1	**Candidate 2**
Higher Overlapping Irradiated Area	Higher Disinfected Area (more 0.22%)
Equal Radiation Exposure Time	Less Disinfection Time (less +/- 3 min)
Equal Intensity of UV radiation per Square Meter	Equal Radiation Exposure Time
	Equal Intensity of UV radiation per Square Meter

Table 6.8: Comparison of the two final candidates' configurations advantages.

Analyzing the Table 6.8, we verify that both candidates present not only equal intensity of UV radiation per square meter but also equal radiation exposure time, so that these metrics are not helpful in the final decision. Candidate 2 takes approximately 3 minutes more disinfection time than candidate 1, which depending on the space disinfected can be significant or not. The difference of percentage of disinfected area between the two candidates is 0.29%, which means that this metric won't be decisive in the final choice either. Regarding the overlapping of irradiated area, candidate 1 achieves a better result due to having a lower `tool_radius` than candidate 2, which successively means that candidate 2 is able to cover the same area more than once, enabling more effective disinfection of the space.

Our final choice was to give more weight to the overlapping of the irradiated area rather than disinfection time, leaving us to opt by the candidate 1. Figure 6.4 shows the state of the environment tested after disinfection using the values of the parameters of the candidate 1.

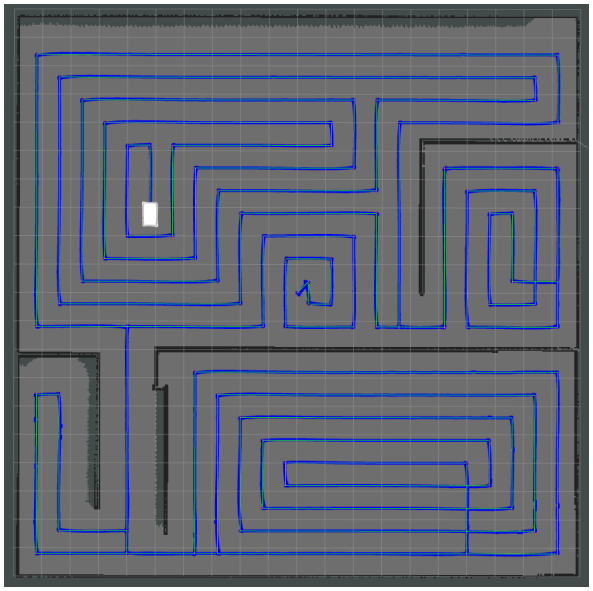

Figure 6.4: Environment state after disinfection.

To conclude, we could obtain a 97.10% of disinfected area in 22 minutes and 19 seconds with some of the area irradiated more than once. A method to calculate the value of this area is to multiply the resolution of the map by the number of pixels that were irradiated more than once. Unfortunately, this would require the usage of a different type of occupancy map. The mapping coverage was made successfully leaving only some areas close to the walls without disinfection, which it is caused by the way BSA algorithm used computes the path.

CHAPTER 7

Conclusion and future work

7.1 Conclusions

The main goal of this work was to develop a system that is able to disinfect an indoor space using the robot MiR100 alongside a UV Sensor as a method of disinfection. We began by researching existing mobile navigation methods for space coverage and space exploration. Then, a survey of existing disinfection robots that used an UV Sensor as a method of disinfection was done. This study helped us understand what method could be used or adapted to our needs in order to be able to cover an entire indoor space. Finally, we chose the researched coverage path planner that was more suited according to all the MiR100's tools available, using the requirements of the proposed robot navigation for environmental disinfection as a reference frame. We chose the LaMa localization algorithm, BSA as coverage planner, `tracking_pid` as local planner and Gazebo as simulation environment. Simulating the UV sensor, a critical component of this project, was a challenging task that was successfully achieved.

In conclusion, the developed simulated disinfection system proved to be working in an a priori known static environment. The robot is able to cover and disinfect an environment at the same time, which help us test different characteristics of the UV Sensor.

7.2 Future work

One such area involves the integration of local planners with collision avoidance. This capability would extend our application of robots to crowded or dynamically changing spaces, including public areas, warehouses, and healthcare facilities. Furthermore, achieving complete coverage of designated areas is essential to obtaining a more efficient disinfection. After the initial `full_coverage_path_planning`, implementing a wall follower algorithm proves to be invaluable. This algorithm would irradiate areas next to walls, guaranteeing a thorough disinfection process. Its integration significantly enhances the efficiency and effectiveness of our system.

Developing a simulation of the UV sensors by incorporating real-world physics and creating intricate 3D models would increase accuracy. This enhancement ensures a realistic simulation

environment, enabling precise testing and validation of the robot's behavior under diverse conditions. Moreover, implementing visualization methods that display areas irradiated more than once would contribute to obtaining crucial insights about the state of disinfection. These visualizations would not only help in identifying redundant coverage but also would allow for adjustments in the robot's behavior.

In conclusion, addressing these challenges would contribute to a more reliable, efficient, and adaptable system that is able to disinfect an indoor space using a UV Sensor as a method of disinfection.

Bibliography

[1] Roland Siegwart and Illah R. Nourbakhsh and Davide Scaramuzza, *Introduction to Autonomous Mobile Robots*. MIT press, 2011.

[2] Shoudong Huang and Gamini Dissanayake, *Robot Localization: An Introduction*. 2016, 1–10. DOI: https://doi.org/10.1002/047134608X.W8318.

[3] Prabin Kumar Panigrahi and Sukant Kishoro Bisoy, «Localization strategies for autonomous mobile robots: A review», *Journal of King Saud University - Computer and Information Sciences*, vol. 34, 6019–6039, 8 Sep. 2022. DOI: 10.1016/j.jksuci.2021.02.015.

[4] Dieter Fox and Wolfram Burgard and Sebastian Thrun and Armin B Cremers, «Position Estimation for Mobile Robots in Dynamic Environments», *American Association for Artificial Intelligence (AAAI/IAAI)*, 983–988, 1998. [Online]. Available: https://www.aaai.org/Papers/AAAI/1998/AAAI98-139.pdf.

[5] Welch, Greg and Bishop, Gary and others, «An introduction to the Kalman filter», 1995. [Online]. Available: https://perso.crans.org/club-krobot/doc/kalman.pdf.

[6] M. Sanjeev Arulampalam and Simon Maskell and Neil Gordon and Tim Clapp, «A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking», *IEEE Transactions on Signal Processing*, vol. 50, 174–188, 2 2002. DOI: 10.1109/78.978374.

[7] Dieter Fox and Wolfram Burgard and Frank Dellaert and Sebastian Thrun, «Monte Carlo Localization: Efficient Position Estimation for Mobile Robots», 1999, 343–349. [Online]. Available: https://www.aaai.org/Papers/AAAI/1999/AAAI99-050.pdf.

[8] Frank Dellaert and Dieter Fox and Wolfram Burgard and Sebastian Thrun, «Monte Carlo Localization for Mobile Robots», 1999, 1322–1328. DOI: 10.1109/ROBOT.1999.772544.

[9] Sebastian Thrun and Wolfram Burgard and Dieter Fox, *Probabilistic Robotics*. MIT Press, 2006.

[10] Gutmann, Jens-Steffen and Schlegel, Christian, «AMOS : comparison of scan matching approaches for self-localization in indoor environments», in *Proceedings of the First Euromicro Workshop on Advanced Mobile Robots (EUROBOT '96)*, IEEE, 1996, 61–67. DOI: 10.1109/EURBOT.1996.551882.

[11] Eurico Pedrosa and Artur Pereira and Nuno Lau, «Efficient localization based on scan matching with a continuous likelihood field», Institute of Electrical and Electronics Engineers Inc., Jun. 2017, 61–66. DOI: 10.1109/ICARSC.2017.7964053.

[12] Piotr Skrzypczynski, «Mobile Robot Localization: Where We Are and What Are the Challenges?», Roman Szewczyk and Cezary Zieliński and Małgorzata Kaliczyńska, Ed., vol. 550, Springer International Publishing, 2017, 249–267. DOI: 10.1007/978-3-319-54042-9.

[13] Dieter Fox, «KLD-Sampling: Adaptive Particle Filters», MIT Press, 2001. [Online]. Available: https://proceedings.neurips.cc/paper/2001/file/c5b2cebf15b205503560c4e8e6d1ea78-Paper.pdf.

[14] Antoni Burguera and Gabriel Oliver and Juan D. Tardos, «Robust scan matching localization using ultrasonic range finders», 2005, 1367–1372. DOI: 10.1109/IROS.2005.1545183.

[15] Sam T. Pfister and Kristo L. Kriechbaum and Stergios I. Roumeliotis and Joel W. Burdick, «Weighted range sensor matching algorithms for mobile robot displacement estimation», vol. 2, 2002, 1667–1674. DOI: 10.1109/robot.2002.1014782.

[16] G Weiss and E V Puttkamer, «A Map Based on Laserscans Without Geometric Interpretation», 1999. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:hbz:386-kluedo-2857.

[17] F Lu and E Milios, «Globally Consistent Range Scan Alignment for Environment Mapping», *Autonomous Robots*, vol. 4, 333–349, 1997. DOI: https://doi.org/10.1023/A:1008854305733.

[18] Luis Montesano and Javier Minguez and Luis Montano, «Probabilistic scan matching for motion estimation in unstructured environments», IEEE Computer Society, 2005, 3499–3504. DOI: 10.1109/IROS.2005.1545182.

[19] Andrea Censi, «An ICP variant using a point-to-line metric», IEEE Xplore, 2008, 19–25. DOI: 10.1109/ROBOT.2008.4543181.

[20] Antoni Burguera and Yolanda González and Gabriel Oliver, «On the use of likelihood fields to perform sonar scan matching localization», *Autonomous Robots*, vol. 26, 203–222, May 2009. DOI: 10.1007/s10514-009-9108-0.

[21] David H. Douglas and Thomas K. Peucker, «Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature», *Cartographica: the international journal for geographic information and geovisualization*, vol. 10, 112–122, 1973. DOI: 10.3138/FM57-6770-U75U-7727.

[22] Todd K. Moon, «The Expectation-Maximization Algorithm», *IEEE Signal Processing Magazine*, vol. 13, 47–60, 1996. DOI: 10.1109/79.543975.

[23] Martin A Fischler and Robert C Bolles, «Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography», *Commun. ACM*, vol. 24, 381–395, 1981. DOI: 10.1145/358669.358692.

[24] J Illingworth and J Kittler, «A Survey of the Hough Transform», *Computer Vision, Graphics, and Image Processing*, vol. 44, 87–116, 1988. DOI: 10.1016/S0734-189X(88)80033-1.

[25] Ali Siadat and Axel Kaske and Siegfried Klausmann and Michel Dufaut and René Husson, «An Optimized Segmentation Method for a 2D Laser-Scanner Applied to Mobile Robot Navigation», *IFAC Proceedings Volumes*, vol. 30, 149–154, 1997. DOI: 10.1016/S1474-6670(17)43255-1.

[26] L. Zhang and B.K. Ghosh, «Line segment based map building and localization using 2D laser rangefinder», vol. 3, IEEE, 2000, 2538–2543. DOI: 10.1109/ROBOT.2000.846410.

[27] Viet Nguyen and Stefan Gächter and Agostino Martinelli and Nicola Tomatis and Roland Siegwart, «A comparison of line extraction algorithms using 2D range data for indoor mobile robotics», *Autonomous Robots*, vol. 23, 97–111, Aug. 2007. DOI: 10.1007/s10514-007-9034-y.

[28] Jan Elseberg and Ross T. Creed and Rolf Lakaemper, «A line segment based system for 2D global mapping», 2010, 3924–3931. DOI: 10.1109/ROBOT.2010.5509138.

[29] Benjamin J. Kuipers and Yung-Tai Byun, «A Robust, Qualitative Method for Robot Spatial Learning», 1988, 774–779. [Online]. Available: https://www.cs.utexas.edu/users/ai-lab/pubs/Kuipers+Byun-aaai-88.pdf.

[30] Howie Choset and Keiji Nagatani, «Topological simultaneous localization and mapping (SLAM): Toward exact localization without explicit localization», *IEEE Transactions on Robotics and Automation*, vol. 17, 125–137, Apr. 2001. DOI: 10.1109/70.928558.

[31] Hongtai Cheng and Heping Chen and Yong Liu, «Topological Indoor Localization and Navigation for Autonomous Mobile Robot», *IEEE Transactions on Automation Science and Engineering*, vol. 12, 729–738, Apr. 2015. DOI: 10.1109/TASE.2014.2351814.

[32] Yao Qi and Rendong Wang and Binbing He and Feng Lu and Youchun Xu, «Compact and Efficient Topological Mapping for Large-Scale Environment with Pruned Voronoi Diagram», *Drones*, vol. 6, Jul. 2022. DOI: 10.3390/drones6070183.

[33] Wolfram Burgard and Martial Hebert and Maren Bennewitz, *World Modeling*. Springer Berlin Heidelberg, 2008, 853–869. DOI: 10.1007/978-3-540-30301-5_37.

[34] Rafael Murrieta-Cid and Carlos Parra and Michel Devy, «Visual Navigation in Natural Environments: From Range and Color Data to a Landmark-Based Model», *Autonomous Robots*, vol. 13, 143–168, 2002. DOI: 10.1023/A:1019685425452.

[35] Sebastian Thrun, «Robotic Mapping: A Survey», *Exploring artificial intelligence in the new millennium*, vol. 1, 1–35, 2002. [Online]. Available: http://ftp.itam.mx/pub/alfredo/ROBOTICS/Navegacion/thrun.pdf.

[36] Adam Milstein, «Occupancy Grid Maps for Localization and Mapping», *Mobile Robots Motion Planning, New Challenges*, 381–408, 2008.

[37] Cyrill Stachniss, *Robotic Mapping and Exploration*, Bruno Siciliano and Oussama Khatib and Frans Groen, Ed. Springer Berlin Heidelberg, 2009, vol. 55. DOI: 10.1007/978-3-642-01097-2.

[38] E. W. Dijkstra, «A Note on Two Problems in Connexion with Graphs», *Numerische Mathematik*, vol. 1, 269–271, 1959. DOI: 10.1007/BF01386390.

[39] Sá, André, «Navigation of autonomous mobile robots», PhD thesis, 2017. [Online]. Available: http://hdl.handle.net/10773/23832.

[40] Peter E. Hart and Nils J. Nilsson and Bertram Raphael, «A Formal Basis for the Heuristic Determination of Minimum Cost Paths», *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, 100–107, 1968. DOI: 10.1109/TSSC.1968.300136.

[41] Stefan Edelkamp and Shahid Jabbar and Alberto Lluch Lafuente, «Cost-Algebraic Heuristic Search», 2005, 1362–1367. [Online]. Available: https://tinyurl.com/chjb4dmt.

[42] Maxim Likhachev and Geoff Gordon and Sebastian Thrun, «ARA*: Anytime A* with Provable Bounds on Sub-Optimality», 2003, 767–774. [Online]. Available: https://tinyurl.com/bdcn3shh.

[43] Maxim Likhachev and Dave Ferguson and Geoff Gordon and Anthony Stentz and Sebastian Thrun, «Anytime Dynamic A*: An Anytime, Replanning Algorithm», 2005, 262–271. [Online]. Available: http://ri.cmu.edu/pub_files/pub4/likhachev_maxim_2005_1/likhachev_maxim_2005_1.pdf.

[44] Maxim Likhachev and Dave Ferguson, «Planning long dynamically feasible maneuvers for autonomous vehicles», *International Journal of Robotics Research*, vol. 28, 933–945, Aug. 2009. DOI: 10.1177/0278364909340445.

[45] Mihail Pivtoraiko and Alonzo Kelly, «Generating Near Minimal Spanning Control Sets for Constrained Motion Planning in Discrete State Spaces», 2005, 3231–3237. DOI: 10.1109/IROS.2005.1545046.

[46] Thomas M. Howard and Alonzo Kelly, «Optimal rough terrain trajectory generation for wheeled mobile robots», *International Journal of Robotics Research*, vol. 26, 141–166, Feb. 2007. DOI: 10.1177/0278364906075328.

[47] Y. Gabriely and E. Rimon, «Spanning-tree based coverage of continuous areas by a mobile robot», vol. 2, Institute of Electrical and Electronics Engineers Inc., 2001, 1927–1933. DOI: 10.1109/robot.2001.932890.

[48] Yoav Gabriely and Elon Rimon, «Spiral-STC: An on-line coverage algorithm of grid environments by a mobile robot», vol. 1, 2002, 954–960. DOI: 10.1109/ROBOT.2002.1013479.

[49] ——, «Competitive on-line coverage of grid environments by a mobile robot», *Computational Geometry*, vol. 24, 197–224, 2003. DOI: 10.1016/S0925-7721(02)00110-4.

[50] Enrique González and Mauricio Alarcón and Paula Aristizábal and Carlos Parra, «BSA: A Coverage Algorithm», vol. 2, 2003, 1679–1684. DOI: 10.1109/iros.2003.1248885.

[51] Enrique González and Oscar Álvarez and Yul Díaz and Carlos Parra and Cesar Bustacara, «BSA: A complete coverage algorithm», vol. 2005, 2005, 2040–2044. DOI: 10.1109/ROBOT.2005.1570413.

[52] Fox Dieter and Wolfram Burgard and Sebastian Thrun, «The Dynamic Window Approach to Collision Avoidance», *IEEE Robotics & Automation Magazine*, vol. 4, 23–33, 1997. DOI: 10.1109/100.580977.

[53] Christoph Rösmann and Wendelin Feiten and Thomas Wösch and Frank Hoffmann and Torsten Bertram, «Trajectory modification considering dynamic constraints of autonomous robots», 2012, 1–6. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6309484.

[54] Christoph Rosmann and Wendelin Feiten and Thomas Wosch and Frank Hoffmann and Torsten Bertram, «Efficient trajectory optimization using a sparse model», IEEE Computer Society, 2013, 138–143. DOI: 10.1109/ECMR.2013.6698833.

[55] Sean Quinlan and Oussama Khatib, «Elastic bands: Connecting path planning and control», vol. 2, Publ by IEEE, 1993, 802–807. DOI: 10.1109/robot.1993.291936.

[56] Pedrosa, Eurico and Pereira, Artur and Lau, Nuno, «A sparse-dense approach for efficient grid mapping», in *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 2018, 136–141. DOI: 10.1109/ICARSC.2018.8374173.

[57] ——, «A Non Linear Least Squares Approach to SLAM using a Dynamic Likelihood Field», *Journal of Intelligent Robotic Systems*, vol. 93, Mar. 2019. DOI: 10.1007/s10846-017-0763-7.

[58] ——, «Fast Grid SLAM Based on Particle Filter with Scan Matching and Multithreading», in *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 2020, 194–199. DOI: 10.1109/ICARSC49921.2020.9096191.

[59] YoonSeok Pyo and HanCheol Cho and RyuWoon Jung and TaeHoon Lim, *ROS Robot Programming: From the basic concept to practical programming and robot application*. ROBOTICS Co.,Ltd., 2017.

[60] Susanne A. Kraemer and Arthi Ramachandran and Gabriel G. Perron, «Antibiotic pollution in the environment: From microbial ecology to public policy», *Microorganisms*, vol. 7, 6 Jun. 2019. DOI: 10.3390/microorganisms7060180.

[61] Da Hu and Hai Zhong and Shuai Li and Jindong Tan and Qiang He, «Segmenting areas of potential contamination for adaptive robotic disinfection in built environments», *Building and Environment*, vol. 184, 10 2020, 15 citações. DOI: 10.1016/j.buildenv.2020.107226.

[62] Xi Vincent Wang and Lihui Wang, «A literature survey of the robotic technologies during the COVID-19 pandemic», *Journal of Manufacturing Systems*, vol. 60, 823–836, Jul. 2021. DOI: 10.1016/j.jmsy.2021.02.005.

[63] Yang Shen and Dejun Guo and Fei Long and Luis A. Mateos and Houzhu Ding and Zhen Xiu and Randall B. Hellman and Adam King and Shixun Chen and Chengkun Zhang and Huan Tan, «Robots under COVID-19 Pandemic: A Comprehensive Survey», *IEEE Access*, vol. 9, 1590–1615, 2021. DOI: 10.1109/ACCESS.2020.3045792.

[64] Robin R. Murphy and Vignesh Babu Manjunath Gandudi and Justin Adams, «Applications of Robots for COVID-19 Response», *arXiv preprint arXiv:2008.06976*, Aug. 2020, 45 citações. DOI: 10.48550/arXiv.2008.06976.

[65] Yu Lin Zhao and Han Pang Huang and Tse Lun Chen and Pen Chi Chiang and Yi Hung Chen and Jiann Horng Yeh and Chien Hsien Huang and Ji Fan Lin and Wei Ting Weng, «A Smart Sterilization Robot System with Chlorine Dioxide for Spray Disinfection», *IEEE Sensors Journal*, vol. 21, 22047–22057, 19 10 2021. DOI: 10.1109/JSEN.2021.3101593.

[66] Stepan Perminov and Nikita Mikhailovskiy and Alexander Sedunin and Iaroslav Okunevich and Ivan Kalinov and Mikhail Kurenkov and Dzmitry Tsetserukou, «UltraBot: Autonomous Mobile Robot for Indoor UV-C Disinfection», IEEE Computer Society, 2021, 2147–2152. DOI: 10.1109/CASE49439.2021.9551413.

[67] Hess, Wolfgang and Kohler, Damon and Rapp, Holger and Andor, Daniel, «Real-time loop closure in 2D LIDAR SLAM», in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, 1271–1278. DOI: 10.1109/ICRA.2016.7487258.

[68] Kaicheng Ruan and Zehao Wu and Qingsong Xu, «Smart cleaner: A new autonomous indoor disinfection robot for combating the covid-19 pandemic», *Robotics*, vol. 10, 3 Sep. 2021, 13 citações. DOI: 10.3390/robotics10030087.

[69] Yunzhou Fan and Yu Hu and Li Jiang and Qian Liu and Lijuan Xiong and Jing Pan and Wenlu Hu and Yao Cui and Tingting Chen and Qiang Zhang, «Intelligent disinfection robots assist medical institutions in controlling environmental surface disinfection», *Intelligent Medicine*, vol. 1, 19–23, 1 May 2021, 3 citações. DOI: 10.1016/j.imed.2021.05.004.

[70] Mur-Artal, Raúl and Tardós, Juan D., «Visual-Inertial Monocular SLAM With Map Reuse», *IEEE Robotics and Automation Letters*, vol. 2, no. 2, 796–803, 2017. DOI: 10.1109/LRA.2017.2653359.

[71] Jing Pan and Yixing Feng and Man Shen and Vanadium Zhi Su and Liang Liu and Tao Xu, «Indoor global positioning method for service robots», CN 106052693 B, 2019.

[72] ——, «Method for fusion of multiple time-of-flight sensors for service robots», CN 105955272 B, 2019.

[73] «Environmental Germicidal Inactivation Efficacy of a UVD Robot in an operating theater/room (ot/or) and traumatology and orthopedic department, general hospital "Dr. Ivo Pedišić" Sisak, Croatia», UVD Robots, 2021.

[74] «Ava UV Disinfection Robot: Promoting a Safer Workspace», Ava Robotics. [Online]. Available: https://www.avarobotics.com/_files/ugd/90f1de_e18953a756b74d08a783324587e3acb3.pdf.

[75] Enric Galceran and Marc Carreras, «A survey on coverage path planning for robotics», *Robotics and Autonomous Systems*, vol. 61, 1258–1276, 12 12 2013. DOI: 10.1016/j.robot.2013.09.004.

[76] Ercan U Acar and Howie Choset and Alfred A Rizzi and Prasad N Atkar and Douglas Hull, «Morse Decompositions for Coverage Tasks», *The International Journal of Robotics Research*, vol. 21, 331–344, 2002. DOI: 10.1177/027836402320556359.

[77] Jiří Hörner, «Map-merging for multi-robot system», Faculty of Mathematics and Physics, 2016. [Online]. Available: https://dspace.cuni.cz/bitstream/handle/20.500.11956/83769/BPTX_2015_1_11320_0_410161_0_174125.pdf?sequence=1&isAllowed=y.

[78] Mobile Industrial Robots A/S, *MiR100 User Guide*, 2020. [Online]. Available: https://www.mobile-industrial-robots.com/solutions/robots/mir100/.

[79] Brodskiy, Yury and Schoenmakers, Ferry and Clephas, Tim and Unkel, Jerrel and van Beek, Loy and Lopez, Cesar, *Full Coverage Path Planner (FCPP)*, 2020. [Online]. Available: https://github.com/nobleo/full_coverage_path_planner.

[80] Franke, Michiel and Lopez, Cesar, *Tracking PID*, 2020. [Online]. Available: https://github.com/nobleo/tracking_pid.