



LOURENÇO AIRES DE SOUSA SILVEIRA RODRIGUES
Master in Electrical and Computer Engineering

AN AUTONOMOUS NAVIGATION SYSTEM FOR COST-EFFECTIVE MOBILE DISINFECTION ROBOTS

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING
NOVA University Lisbon
<September>, <2023>



AN AUTONOMOUS NAVIGATION SYSTEM FOR COST-EFFECTIVE MOBILE DISINFECTION ROBOTS

LOURENÇO AIRES DE SOUSA SILVEIRA RODRIGUES

Master in Electrical and Computer Engineering

Adviser: José António Barata Oliveira

Full Professor, NOVA University Lisbon

Co-adviser: Francisco Antero Cardoso Marques

Research Engineer, NOVA University Lisbon

An Autonomous Navigation System for Cost-Effective Mobile Disinfection Robots

Copyright © Lourenço Aires de Sousa Silveira Rodrigues, NOVA School of Science and Technology, NOVA University Lisbon.

The NOVA School of Science and Technology and the NOVA University Lisbon have the right, perpetual and without geographical boundaries, to file and publish this dissertation through printed copies reproduced on paper or on digital form, or by any other means known or that may be invented, and to disseminate through scientific repositories and admit its copying and distribution for non-commercial, educational or research purposes, as long as credit is given to the author and editor.

ABSTRACT

The Covid-19 pandemic evinced that some viruses can rapidly spread and infect a lot of people and that, in these cases, the need to disinfect becomes urgent. The pandemic also emphasized the importance of conducting disinfection procedures in a safe, fast, and sustainable way.

Autonomous mobile robots are starting to be part of our daily reality; there are already robots cleaning houses, serving tables, and helping in warehouses. Robots allow humans to concentrate on complex tasks that need continuous adaptation and human judgment, while robots perform repetitive, tedious, and sequential tasks.

This dissertation focuses on achieving safer, faster, and more sustainable disinfection using UV-C radiation lights in autonomous mobile robots. It does so while guaranteeing this project has the lowest budget possible in order to be accessible to smaller businesses. The latter will be able to not only quickly disinfect hospital rooms and corridors, but will also protect humans from contacting with various diseases as they disinfect multiple rooms. It will do so in a much more sustainable way, as UV-C lights can be used for over 12000 hours, compared to single-use materials like gloves, cloths, and masks needed to disinfect the rooms.

In this dissertation, a system is developed with four major blocks: Map, Algorithm, Navigation, and Hardware. This system has the ability to map a floor, determine where the robot should stop to turn on the disinfection lights and guide the robot through those stopping points. This enables the disinfection of parts of the environment such as rooms, corridors, operation rooms, etc.

Keywords: disinfection, UV-C radiation, autonomous mobile robots, ROS2, autonomous navigation, path planning.

RESUMO

A pandemia de Covid-19 demonstrou que alguns vírus podem propagar-se rapidamente e infectar muitas pessoas e que, nestes casos, a necessidade de desinfetar se torna urgente. A pandemia também enfatizou a importância de realizar procedimentos de desinfecção de forma segura, rápida e sustentável.

Os robôs móveis autónomos começam a fazer parte da nossa realidade diária; já existem robôs a limpar casas, a servir mesas e a ajudar em armazéns. Os robôs permitem que os humanos se concentrem em tarefas complexas que necessitam de adaptação contínua e de julgamento humano, enquanto os robôs executam tarefas repetitivas, entediantes e sequenciais.

Esta dissertação foca-se em conseguir uma desinfecção mais segura, mais rápida e mais sustentável utilizando luzes de radiação UV-C em robôs móveis autónomos. Mantendo este projeto com o orçamento mais baixo possível, a fim de ser acessível a pequenas empresas. Estes serão capazes não só de desinfetar rapidamente quartos e corredores de hospitais, mas também de proteger os seres humanos do contacto com várias doenças enquanto desinfetam múltiplos quartos. Fazer-lo-á de uma forma muito mais sustentável, uma vez que as luzes UV-C podem ser utilizadas durante mais de 12000 horas, em comparação com os materiais de utilização única, como luvas, panos e máscaras, necessários para desinfetar os quartos.

Nesta dissertação, é desenvolvido um sistema com quatro grandes blocos: Mapa, Algoritmo, Navegação e Hardware. Este sistema tem a capacidade de mapear um piso, determinar onde o robot deve parar para acender as luzes de desinfecção e guiar o robot através desses pontos de paragem. Isto permite a desinfecção de partes do ambiente, tais como quartos, corredores, salas de operações, etc.

Palavras-chave: desinfecção, radiação UV-C, robôs móveis autónomos, ROS2, navegação autónoma, planeamento de trajetórias.

CONTENTS

List of Figures	7
Glossary	11
Acronyms	13
1 Introduction	1
1.1 Context	1
1.2 Problem	2
1.3 Proposed Solution	3
2 State of the Art	5
2.1 Robots	5
2.1.1 Mobile Robots	5
2.2 Path Planning	6
2.2.1 Global Planning	6
2.2.2 Local Planning	8
2.3 Human-aware navigation	11
2.4 Ultraviolet-C radiation	15
3 Supporting Concepts	17
3.1 Robot Operating System (ROS)	17
3.1.1 Nodes	18
3.1.2 Topics	19
3.1.3 Services	20
3.1.4 Parameters	20
3.1.5 Actions	21
3.2 SLAM	23
4 Proposed Model	24
4.1 Map	25

4.2	Planner	26
4.3	Navigation	30
4.4	Platform	31
5	Implementation	32
5.1	Platform	33
5.2	Navigation	36
5.3	Planner	38
5.4	Map	40
6	Experimental Results	42
6.1	Turtlebot3	42
6.2	Gazebo World	47
7	Conclusions and Future Work	52
7.1	Conclusions	52
7.2	Future Work	53
	Bibliography	54

LIST OF FIGURES

1.1	Robotic Arm used for car manufacturing	2
1.2	UV-C Application examples	3
1.3	UV-C Mobile Robot example [8]	4
2.1	Mobile Robots examples	6
2.2	Dijkstra Algorithm example with nodes	7
2.3	Comparison between the two algorithms [13]	8
2.4	Traces of the robot's odometry and the humans' movements (solid gray lines). Cell size: 1m^2 [16]	10
2.5	Translational velocity profile of the robot [16]	11
2.6	First test: (a) simulated physical therapy room with three objects inside. Time- dependent social interaction spaces and their costs in the graph: (b) yellow, no activities scheduled; (c) red, activities are being performed; and (d) orange, activities have just finished. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)[17].	12
2.7	Birds eye view of the first environment. The participants (pink trajectory) moved from a sofa to a table, while observing one of two trajectories, which corresponds to the robot behavior resulting from the author's previous work [19] (orange trajectory) and the approach presented in this paper (green tra- jectory). Using the new approach, the robot avoids being visible as much as possible [18].	13
2.8	The author's system deployed and tested on a real robot, Pepper. The top pictures show the open space scenario and the bottom pictures show the test of the Backoff-recovery mode [22].	15
3.1	ROS node example, based on [28]	18
3.2	ROS topic example, based on [29]	19
3.3	ROS service example, based on [30]	20
3.4	ROS action example, goal request, based on [32]	21
3.5	ROS action example, goal response, based on [32]	21

3.6	ROS action example, result request, based on [32]	22
3.7	ROS action example, feedback, based on [32]	22
3.8	ROS action example, result response, based on [32]	22
3.9	Robot using SLAM Toolbox to map an environment	23
4.1	Proposed Model	24
4.2	Hospital floor plan	25
4.3	Two different layouts of hospital room 1 - Render	26
4.4	Two different layouts of hospital room 1 - Map	26
4.5	Disinfection lights radius	27
4.6	Hospital floor plan with POIs	27
4.7	Hospital floor plan with POIs and path between them	28
4.8	Hospital rooms with different layouts and paths	29
4.9	Hospital room with different layout and path blocked	29
4.10	Navigation	30
4.11	Autonomous Mobile Robot examples	31
5.1	Implementation Block scheme	32
5.2	Implementation Block scheme - Platform	33
5.3	Tracer by Agilex [35]	33
5.4	Tracer by Agilex - Top view [35]	34
5.5	Hokuyo UTM-30LX Scanning Laser Rangefinder [38]	34
5.6	Philips TUV Amalgam XPT [39]	35
5.7	Robot under construction	35
5.8	Implementation Block scheme - Navigation	36
5.9	Nav2 architecture [40]	37
5.10	Implementation Block scheme - Planner	38
5.11	Turtlebot3 Simulation. Representation of the waypoints distribution. Waypoints in red, walls of the room and obstacles in black, free space in white, and unknown space in light grey	39
5.12	Turtlebot3 Simulation. Representation of the path made by the planner. Path in red, walls of the room and obstacles in black, free space in white, and unknown space in light grey	39
5.13	Turtlebot3 Simulation. Representation of a path between two waypoints planned by the Nav2 navigation. Walls are represented in light purple, free space is represented in white, light blue is space where the robot cannot go because of its size, and a gradient of colors, from dark purple to dark red, identifies from the free zone to the forbidden zone respectively. There are also the blue, green, and red lines that identify the axis from the robot, the green thin line that is the distance to the Transform Framework (TF), and the pink path with a small arrow is the path created by the Nav2 navigation.	40

5.14 Implementation Block scheme - Map	40
5.15 Turtlebot3 simulation. Representation of the robot mapping using SLAM Toolbox. The color code was explained before, but adding to that, there is a more intense colored square that is what the LiDAR sensor is detecting at the moment, the lighter colors are part of the map that is already created, and the green square is the size of the map at this given time.	41
5.16 Turtlebot3 simulation. Representation of the Map created by SLAM Toolbox. The walls of the room and obstacles are represented in black, free space in white, and unknown space in light grey. The numbers on the axis are the pixel scale.	41
6.1 Turtlebot3 simulation. Map with the path to follow. The color code is the same as the previous map created by SLAM Toolbox, in this case, there is a red line which is the path to follow.	43
6.2 Turtlebot3 simulation. Map with waypoints, parameters 0.7m distance between waypoints and 0.2m distance to the walls. The color code is the same as the previous map created by SLAM Toolbox, with the waypoints marked in red.	43
6.3 Turtlebot3 simulation. Map with waypoints, parameters 0.2m distance between waypoints and 0.2m distance to the walls. The color code is the same as the previous map created by SLAM Toolbox, with the waypoints marked in red.	44
6.4 Turtlebot3 simulation. Map with waypoints, parameters 2m distance between waypoints and 0.1m distance to the walls. The color code is the same as the previous map created by SLAM Toolbox, with the waypoints marked in red.	44
6.5 Turtlebot3 simulation. Map with all the waypoints reached. The color code is the same as the figure before that had the information from the LiDAR sensor. In this case, there are the multiple positions of the robot represented by the green circle and the different color lines representing the axis.	45
6.6 Turtlebot3 simulation. The initial map merged with all the waypoints reached. This color code is the same as the figure before, adding the path created by the planner in red.	45
6.7 Turtlebot3 simulation. Two paths that took longer than others to conclude. This figure has the same color code explained before.	46
6.8 Gazebo world. Walls are represented in a lighter grey, the obstacles in a slightly darker grey, the robot is the small cylinder in the middle of the room and the blue area represents the area covered by the LiDAR sensor when the robot is in that position.	47
6.9 Gazebo simulation. Representation of the robot mapping with SLAM Toolbox. This figure has the same color code explained before.	47

6.10 Gazebo simulation. Representation of the Map created by SLAM Toolbox. This figure has the same color code explained before.	48
6.11 Gazebo simulation. Map with the path to follow. This figure has the same color code explained before.	48
6.12 Gazebo simulation. Representation of the path between two waypoints created by Nav2 Navigation. This figure has the same color code explained before.	48
6.13 Gazebo simulation. Map with waypoints, parameters 0.5m distance between waypoints and 0.2m distance to the walls. The color code is the same as the previous map created by SLAM Toolbox, with the waypoints marked in red.	49
6.14 Gazebo simulation. Map with waypoints, parameters 0.7m distance between waypoints and 0.2m distance to the walls. The color code is the same as the previous map created by SLAM Toolbox, with the waypoints marked in red.	49
6.15 Gazebo simulation. Map with waypoints, parameters 0.2m distance between waypoints and 0.2m distance to the walls. The color code is the same as the previous map created by SLAM Toolbox, with the waypoints marked in red.	50
6.16 Gazebo simulation. Map with waypoints, parameters 0.5m distance between waypoints and 0.5m distance to the walls. The color code is the same as the previous map created by SLAM Toolbox, with the waypoints marked in red.	50
6.17 Gazebo simulation. Map with all the waypoints reached. This figure has the same color code explained before.	50
6.18 Gazebo simulation. The initial map merged with all the waypoints reached. This figure has the same color code explained before.	51
6.19 Gazebo simulation. Difficult passage. This figure has the same color code explained before.	51

GLOSSARY

Algorithm	A process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer. 6
Disinfection	The process of killing or removing harmful microorganisms from surfaces or the environment to prevent the spread of diseases. 3
Human-Aware Navigation	A concept in robot navigation where robots are designed to navigate in a socially acceptable way around humans, taking into consideration human presence and comfort. 11
Mobile robots	Robots that have the ability to move autonomously. 1
Navigation	The process of determining a route or path from one location to another, often used in the context of mobile robots. 6
Operating Rooms	Specialized rooms in hospitals where surgical procedures are performed. 4
Pandemic	A global outbreak of a disease, such as COVID-19, affecting a large number of people across multiple countries. 2
Path Planning	The process of finding an optimal path or route for a robot to move from a starting point to a destination while avoiding obstacles. 10

Sustainability	The practice of using resources in a way that does not deplete or harm the environment, ensuring long-term viability. 2
UV-C Radiation Lights	Ultraviolet-C lights used for disinfection, capable of killing bacteria and pathogens on surfaces. 3

ACRONYMS

AGV	Automated Guided Vehicle 33
AMCL	Adaptive Monte Carlo Localization 37
API	Application Programming Interface 37
AUV	Autonomous Underwater Vehicles 5
BT	Behaviour Trees 37
CL	Curriculum Learning 14
DNA	Deoxyribonucleic Acid 15
DRL	Deep Reinforcement Learning 13
DWA	Dynamic Window Approach 10
HATEB	Human-Aware Timed Elastic Band 14
HC	Homology Class 10
HER	Hindsight Experience Replay 14
IoT	Internet of Things 16
LED	Light Emitting Diodes 16
LIDAR	Light Detection and Ranging 23
LMR	Legged Mobile Robots 5
ORCA	Optimal Reciprocal Collision Avoidance 13
POI	Point of Interest 26
PPO	Proximal Policy Optimization 13

RL	Reinforcement Learning 14
ROS	Robot Operating System 5 , 17 , 19 , 21
RS	Reward Shaping 14
SFM	Structure from Motion 13
TEB	Timed Elastic Band 10
TF	Transform Framework 8 , 40
UAV	Unmanned Aerial Vehicles 5
UV-C	Ultraviolet-C 15
UV-B	Ultraviolet-B 15
UV-A	Ultraviolet-A 15
VR	Virtual Reality 12
WMR	Wheeled Mobile Robots 5

INTRODUCTION

1.1 Context

Robots have been around since the early 1950s; we have seen them in factories building our cars (fig.1.1) , moving parts from one place to another, and working in warehouses. In the beginning, in Traditional Industry, robots were only used as factory manipulators. A robot would only do a single task, over and over again, always in the same space, with no collaboration from other robots or humans. The robot would assume that what it was doing was his world, so it couldn't perceive other robots or humans around it. For example, if a human would put his arm in the range of the robot when it was moving, the robot would collide with the human's arm and wouldn't even know what it was or how it affected the work that the robot was doing. The robot could only be used for one task, i.e. in a car factory a robot that tightened screws would only do that, it wasn't able, i.e. to change a screwdriver for a claw to grab something after tightening the screws. Even after some years when [Mobile robots](#) started to appear they would only navigate in warehouses where there were no humans moving around or where there were lines on the floor they would follow. Robots weren't able to navigate on their own, plan a path, avoid obstacles that weren't pre-defined, or even choose a different path when the previous one was blocked.

Nowadays robots are more and more around us, cleaning our houses, serving tables, driving our cars, working in factories with humans collaborating with them, and moving around them. Having a robot around us can be quite scary, if it is moving too fast people are afraid the robot will bump into them. If the robot is navigating in a certain trajectory and suddenly changes direction, it can intercept our path and make us stumble.



Figure 1.1: Robotic Arm used for car manufacturing

The world is in constant evolution, humans are always looking for better and faster ways to do simple and complex tasks. Since the start of the [Pandemic](#) we have lived in, humans have been even more preoccupied with their well-being, mental health, and [Sustainability](#). In this matter, disinfection has become a keyword and a big concern. Not only because people realized how fast a virus can be spread and how easy it can be to infect another person but also because of all the waste it creates.

1.2 Problem

With the pandemic we lived in, the whole world became aware of how fast a disease can be spread. People became obsessed with disinfection. Some people would change clothes when getting in their homes, some would clean all their groceries with alcohol, and many would use gloves and other means to protect and disinfect the environment around them.

Hospitals, naturally are places where there are a lot of diseases. A place that a lot of people frequent, where a lot of different people from different places with different diseases go to get treatment. With that said, in the pandemic hospitals became a place that a lot of people feared and a place where there were a lot of contaminations.

With this fear and this pandemic, the need to disinfect hospitals and a lot of other public spaces became even more urgent. Not only the need to disinfect but also the need to do it in a safe and sustainable way. As said in [2] "(...) continued efforts to improve the quality and consistency of traditional cleaning and disinfection practices are needed.". It needs to be safe regarding not only the patients, who needed to be sure that the rooms were well disinfected but also the people disinfecting the rooms that were exposed to the diseases. And sustainable because the easiest way to know that something is disinfected is to have single-use materials that after their use are thrown away.

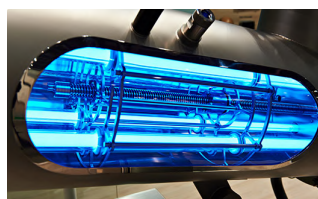
Traditionally, disinfection is done with chemical disinfectants and thorough manual cleaning of surfaces, this method not only is not sustainable but also might not fully disinfect the surfaces. "In the future, the state-of-the-art technologies involving no-touch approaches may be developed to fully disinfect the contaminated surfaces,"[3] this way disinfection can be done in a much safer and more sustainable way.

Also with the pandemic, it became obvious that hospitals were not the only facilities that needed to be seriously disinfected. All environments that have one of two things: either a lot of people who go there every day, like buses, malls, train stations, and airports, or places that are prone to disease, such as clinics, veterinaries, etc., need to be seriously disinfected.

1.3 Proposed Solution

With the problem described before there are different aspects to it. First of all, there is the sustainability matter and the need to ensure that surfaces are completely disinfected. For this matter one of the solutions and the one that is going to be used in this dissertation is the use of [UV-C Radiation Lights](#) in order to disinfect rooms, corridors, halls, and other parts of a hospital. As said before this kind of [Disinfection](#) is also important for other environments, but in this dissertation, the environment that is going to be focused on is the hospital environment.

UV-C Lights in hospitals are a big advantage as the lights reach places that can be difficult for humans to reach, and it can be done in a much faster way. UV-C has been used in a lot of different ways, to disinfect water, air purifiers, mobile devices disinfection, etc. [fig.1.2](#). Studies show that in 15min exposure it kills 99.9% of the bacteria on the surface [\[4\]](#).



(a) UV-C used on water purification [\[5\]](#)



(b) UV-C used on air purification [\[6\]](#)



(c) UV-C used on mobile devices disinfection [\[7\]](#)

Figure 1.2: UV-C Application examples

Regarding the problem of safety and management of people discussed before, having people who are not infected go inside hundreds of different rooms with different diseases is extremely dangerous. That is when robots come in. Having mobile robots, with UV-C lights, navigating autonomously through empty rooms, disinfecting every room, and eliminating 99.9% of bacteria. Even more, having those robots disinfect corridors, elevators, and [Operating Rooms](#), ensures that the hospital is less and less dangerous for those who frequent it.

Today, there are some available solutions with mobile robots and UV-C Lights (fig.1.3), but the problem is that all these solutions are way too expensive. Maybe a big hospital can justify the thousands of euros that one of these robots costs as it has a lot of rooms to disinfect and thousands of patients every day. However, some smaller businesses cannot afford such an expense and still need to have this kind of disinfection. That is the solution that this dissertation is based on: making an autonomous mobile robot that will disinfect with UV-C Radiation Lights the different parts of a hospital with a much lower budget and with a robot that is easy to program allowing it to even be used to different appliances other than disinfecting. The final goal of this dissertation is to test all the frameworks in a real robot with UV-C lights and have some conclusions about the disinfection made by mobile robots with UV-C Lights in a hospital environment.



Figure 1.3: UV-C Mobile Robot example [8]

STATE OF THE ART

2.1 Robots

A robot, is a machine that performs human tasks and much more complicated tasks as well. It is capable of performing tedious tasks over and over again without getting tired or making errors. Some robots can fit in places humans can't, others can operate in environments that humans can't, for instance, extremely hot or in the presence of a toxic gas. Robots give humans the ability to concentrate on more complex tasks, tasks that need continuous adaptation, and tasks that need human judgment, while robots do repetitive, tedious, and sequential tasks.

2.1.1 Mobile Robots

Mobile robots are robots that have the ability to move around autonomously, in contrast to industrial robots that can only operate in a limited range. This ability makes them suitable for larger environments where, for example, they need to move some things from one place to another or where they need to execute tasks in different places. These robots do not need the assistance of an external human operator. Mobile robots can be classified into three different categories (fig.2.1):

1. Ground mobile robots, where there are two types:
 - a) **Wheeled Mobile Robots (WMR)**, these robots are very useful in environments where there are no stairs because they have low energy consumption and are relatively simple in terms of mechanics.
 - b) **Legged Mobile Robots (LMR)**, these robots are used in places where the robot is required to climb stairs, get over obstacles, etc.
2. **Unmanned Aerial Vehicles (UAV)**, these robots are usually controlled by a remote controller but can also be autonomous.
3. **Autonomous Underwater Vehicles (AUV)**, are used mostly to map submerged wrecks, rocks and obstructions that can be a hazard to navigation.



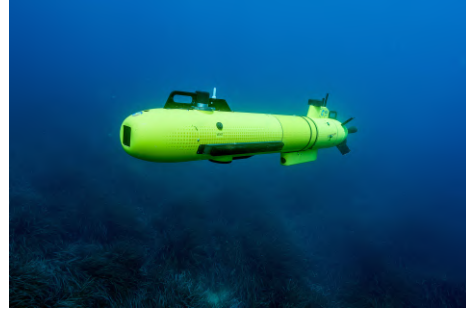
(a) WMR



(b) LMR [9]



(c) UAV



(d) AUV [10]

Figure 2.1: Mobile Robots examples

2.2 Path Planning

Navigation is essential to mobile robots. In order to move from one place to another the robot must locate its starting and ending points, and it must be able to avoid obstacles along the way, to do this the robot needs to plan its path. Path planning can be divided into the global planner, which uses information already known about the environment to plan a path, and the local planner which recalculates the initial plan as the robot starts to follow the path in order to avoid new obstacles.

2.2.1 Global Planning

When planning a path at the starting point, the robot's sensor range is limited, therefore the robot needs to have prior information in order to plan the path before starting to move and achieving a path that is close to the most efficient. In our day-to-day lives the global planner is like planning a trip with a map of the city, (the old ones where there were no traffic indications or accident alerts), with it we can know the best route to take in order to get to the destination we want. In the next sections it is going to be presented two main **Algorithms**, Dijkstra's algorithm and A* Algorithm.

2.2.1.1 Dijkstra's algorithm

Edsger Dijkstra proposed an algorithm [11] that allowed the robot to find the shortest path from a starting point to an ending point, he named it Dijkstra's Algorithm. This algorithm finds the shortest distance between two nodes in a graph, and can therefore, calculate the shortest path when given a starting node and an ending node.

The algorithm starts by creating two separate lists of nodes: the visited nodes list (that begins empty) and the unvisited nodes list. It then sets the distance from the starting node to all other nodes to infinity - as there is no information about that in the beginning - and it sets the distance to the starting node to zero.

After that, it visits the unvisited node with the shortest distance from the starting node, in this case, it is the starting one. In that node, it calculates the distance from the starting node to the nodes connected to the current node. If the calculated distance to the node is less than the recorded distance, then it updates the shortest distance. After that, it updates the previous node for the update's distances. And update the visited list, as well as the unvisited list. Then, it repeats the process starting from visiting the unvisited node with the shortest distance from the starting node, fig.2.2.

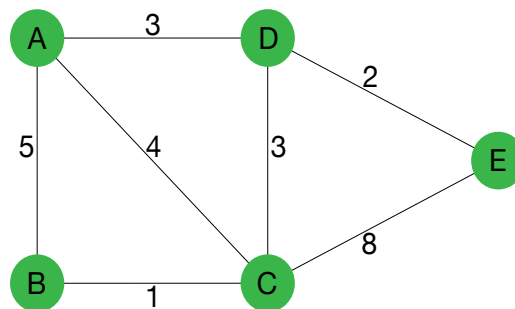


Figure 2.2: Dijkstra Algorithm example with nodes

2.2.1.2 A* Algorithm

A* algorithm, or A Star algorithm, is an algorithm also used to plan the shortest path between two nodes [12]. This algorithm begins by calculating the distance between every two nodes that have a direct link, after that, for each node it is given a value based on its distance in a theoretical straight line to the goal node. Then beginning on the start node, it is calculated the value of $f(n)$, $f(n)$ is the sum of $g(n)$ with $h(n)$ where $g(n)$ is the distance from the starting node and $h(n)$ is the distance to the goal node in a theoretical straight line. Similar to Dijkstra's Algorithm, the value of $f(n)$ is compared with other values of $f(n)$ already calculated, and if it is less the value is updated, as well as the field where the previous node is saved. After this, the node with the smallest $f(n)$ is the next node evaluated and the process is repeated until the goal node is reached.

2.2.1.3 Comparison

Both Dijkstra's Algorithm and A* Algorithm have the ability to find the shortest path from a starting point to an ending point. Comparing the two, as it can be seen in figure 2.3 it can be noticed that Dijkstra's Algorithm takes more time because, from the starting point, it expands all around until the goal node is found. In the A* Algorithm there is only one path that is continued, and that is the one with the smallest $f(n)$ value. With this being said, it doesn't mean that the A* Algorithm is better than Dijkstra's Algorithm. The A* Algorithm tries to take a more direct path, and Dijkstra's Algorithm has a wider search space, but in some cases, it is better because the A* Algorithm assumes that the distance in a straight line is the actual distance when there can be an obstacle in between that makes the assumption wrong.

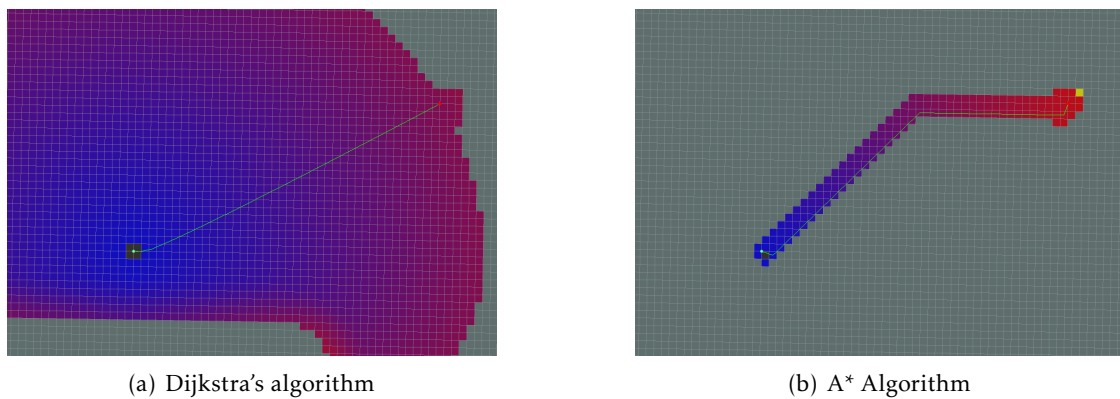


Figure 2.3: Comparison between the two algorithms [13]

2.2.2 Local Planning

When navigating with the help of a map, for example, it is possible to determine the shortest path, or even the fastest one if we take into consideration the speed limit on each road, but it is not possible to predict accidents on the road, cars broken down on the side of the road or even an obstacle like a tree that fell on the road. Like so, in robot navigation, with global planners it is possible to determine the shortest path, but that path doesn't take into consideration new dynamic obstacles that get in the way in the meantime. That is where local planners are needed, local planners are constantly planning a new "shortest path" in order to avoid some new obstacle detected by the sensors on the robot.

2.2.2.1 Timed elastic band

The Timed Elastic Band algorithm is an extension of the Elastic Band algorithm. The Elastic Band approach deforms, in real-time, the global path into a short and smooth path that keeps a safe distance from obstacles. Its objective is to avoid the need to stop and make an aggressive turn from one waypoint to the other, as the global planner would suggest [14]. This approach, like all approaches, has some flaws and so the Timed Elastic Band algorithm is proposed in order to cover some of those flaws. This new approach adds consideration to the robot's kinematic constraints and temporal aspects. It uses the waypoints created by the global planner, to know the global path, and smooths the path in between them, in order to get the fastest path, for that robot, even though it is not the shortest path.

2.2.2.2 Dynamic Window Approach

The authors who wrote [15], proposed a new approach to new dynamic obstacles on the robot's way. Their main goal is to implement a reactive avoidance of collisions with obstacles. Due to the limited velocity and acceleration of the robot, the command to steer has to be given in a short period of time, otherwise, the robot won't have enough time to avoid the obstacles. This approach differs from others because it is derived directly from the motion dynamics of the robot and it takes into consideration the inertia of the robot, which is very important, especially if the robot has torque limits and is traveling at high speeds. This approach has two steps, the search space, and the optimization. The search space of possible velocities can be divided into three steps[15]:

1. Circular trajectories, here the dynamic window only considers circular trajectories.
2. Admissible velocities, this ensure that only safe trajectories are considered. A pair (v,w) is considered admissible if the robot has enough time to stop before reaching the closest obstacle on the corresponding curvature.
3. Dynamic window, this step restricts the admissible velocities to those that can be reached in a short time interval.

The optimization has the goal of maximizing the objective function, considering the current location and orientation of the robot. This function has to balance three aspects:

1. Target heading: heading is what measures the progress towards the goal and it is maximum when the robot moves directly to that position.
2. Clearance: is the distance to the closest obstacle, the smaller the distance is the higher the robot desires to move around it.
3. Velocity: is the forward velocity of the robot and supports fast movements.

2.2.2.3 Homology class

In [16], the authors compare three different local planners, the [Dynamic Window Approach \(DWA\)](#), the [Timed Elastic Band \(TEB\)](#), and the Timed Elastic Band with [Homology Class \(HC\)](#), HC+TEB. The last one hasn't been described in this document and it is one of the main features described in [16]. Homologous paths are two paths that have the same start and end points, avoid the same obstacles but take different intermediate points to make the trajectory. In the HC+TEB approach, homologous paths are kept while the main path is taken. With this, when the robot finds a new obstacle, the whole path won't have to be calculated again and the robot can simply choose a different path in order to avoid the obstacle, this feature enables both forward and backward [Path Planning](#). In fig.2.4, the authors of the article show a scenario where the three approaches are implemented. There it is possible to see that the TEB approach results in a collision with a human, it's preferred path is elongated by the motion of the human starting from the bottom, and the motion of another human approaching from the right results in a collision. Both the DWA and the HC+TEB are able to find a path through the various humans and get to the goal point. The HC+TEB approach finds a path that is very near to the straight line from the beginning point to the goal point, while the DWA got a little further. Fig.2.5 shows the translational velocity profile of the robot for the different approaches and while the DWA needed the robot to slow down in order to avoid collisions, the HC+TEB has the ability to let the robot travel almost at maximum speed the whole way.



Figure 2.4: Traces of the robot's odometry and the humans' movements (solid gray lines). Cell size: 1m^2 [16]

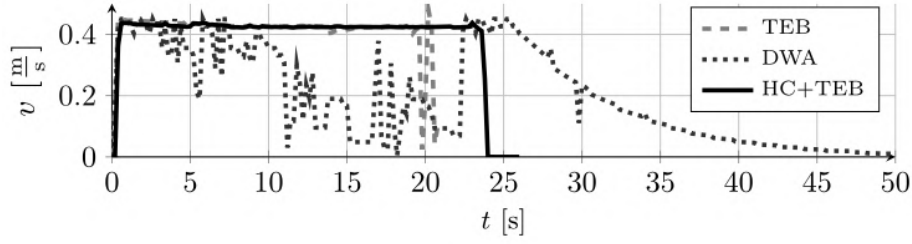


Figure 2.5: Translational velocity profile of the robot [16]

2.3 Human-aware navigation

Human-Aware Navigation is a key element in robot navigation in environments where the robot needs to know how to navigate in a socially acceptable way. Without this element, the robot would navigate while considering humans as static obstacles like furniture and would end up making an ineffective route and likely bother humans around it. In this matter, there are different approaches based on different aspects of the navigation and the environments that the robot will be in.

Some approaches are oriented to adding layers to the robot's navigation, introducing different and important aspects that make humans feel more comfortable around robots.

In [17], it is proposed to use schedules as a way of improving human-aware navigation. What is said, is that time influences humans on a daily basis, even without acknowledging it or unconsciously, time is present in everything that we do and in the way we define our day. Therefore, it is easier to understand that a robot could gain a lot with having a time constraint. In [17], the authors propose the use of a new path layer where time is taken into consideration. In fig.2.6 there is an example of a therapy room that has three objects inside. Applying this to a factory would be something like that. Near a workstation, during work hours there will be actions being performed at the workstation, let's say between 8h and 17h, in this time frame the area near the workstation is going to be red for the robot, so it cannot come near. Between 18h and 7h, that same area will be green so the robot can navigate without those limitations, and between 7h and 8h, 17h, and 18h, and lunchtime the area will be orange because it is not supposed to be anyone there but since it is near the time frame where an operator would be there the robot should be careful. With this framework, the robot can choose different paths varying with the normal schedules for that space, this can help a lot the robot to have a more socially accepted navigation.

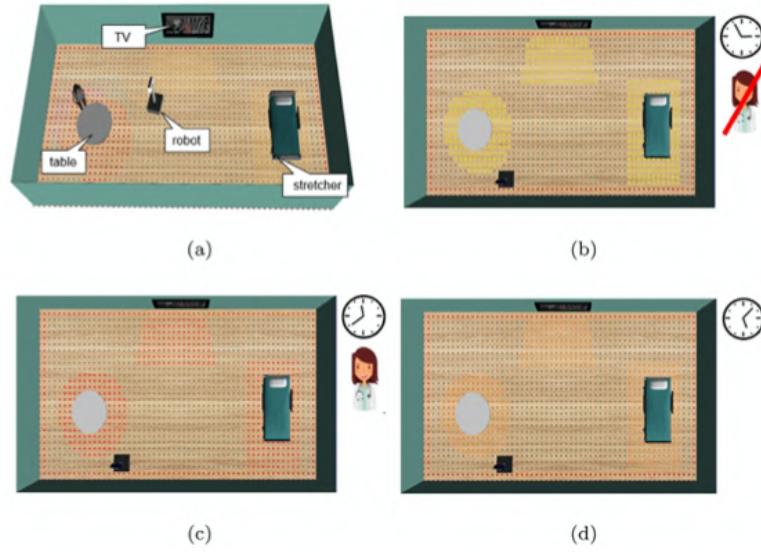


Figure 2.6: First test: (a) simulated physical therapy room with three objects inside. Time-dependent social interaction spaces and their costs in the graph: (b) yellow, no activities scheduled; (c) red, activities are being performed; and (d) orange, activities have just finished. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)[17].

In [18] the authors introduce visibility and predictability as new parameters to the robot's navigation. For this, they studied what makes humans feel more comfortable around a moving robot, they worked with 299 participants in total and came to the conclusion that aside from proximity, humans also value unobtrusiveness and predictability in service robots. The authors thus proposed a weighted cost function, which is used to gather information to add to an A* planner. This allows the robot to choose a path that avoids being visible to the human as much as possible. In a house, for example, if the robot is moving around and there is a human moving in the house too, the less the human sees the robot and the less they run into each other the better, because this will make the human a lot more comfortable being at home with the robot moving around, fig.2.7. After implementing this work, the authors ran some experiments with the participants in a [Virtual Reality \(VR\)](#) environment, and the presented robot navigation approach was perceived positively.

Other approaches are focused in improving the robot navigation in different aspects, like the way that the robot decides which path to take, the way the robot gets its knowledge to know how to avoid humans in an inexpensive way, and even how the same robot can adapt to different environments:

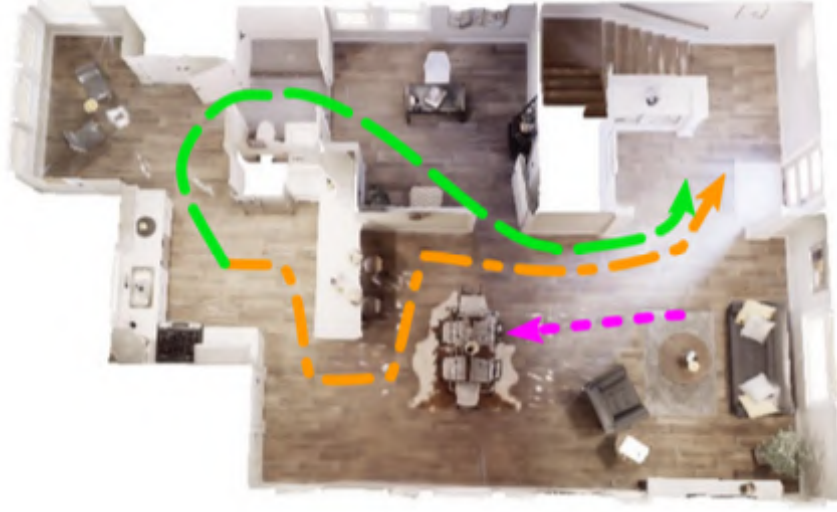


Figure 2.7: Birds eye view of the first environment. The participants (pink trajectory) moved from a sofa to a table, while observing one of two trajectories, which corresponds to the robot behavior resulting from the author’s previous work [19] (orange trajectory) and the approach presented in this paper (green trajectory). Using the new approach, the robot avoids being visible as much as possible [18].

In [20], it is proposed the use of [Deep Reinforcement Learning \(DRL\)](#) for a crowd-aware environment. It allows the robot to recognize and avoid pedestrians that take different paths and collision avoidance strategies. To do that, the authors apply the sensor map and the pedestrian maps as inputs to a neural network. To train the neural network that is going to be used by the robot it is used [Proximal Policy Optimization \(PPO\)](#), proximal policy optimization. PPO uses two different networks during training, a policy network, and a value network. After that, and before being deployed in the robot to use in the real world, the network is trained in simulated environments where pedestrians follow [Optimal Reciprocal Collision Avoidance \(ORCA\)](#) and [Structure from Motion \(SFM\)](#). With this method, the robot is able to distinguish between static objects and various pedestrians and actively avoid both, making the human-aware navigation better, more efficient, and more natural to humans. After various experimentations, the authors came to the conclusion that the proposed approach works well in most cases and that in only a few cases the robot didn’t avoid the pedestrian in front of it. In these cases, it is said that it could be implemented some safety strategy to ensure safety for pedestrians.

In [21], the authors noticed that for most previously used and currently used approaches, there is always a need for some sort of dataset of high-quality human demonstrations in the real-world navigation, to obtain these datasets is expensive and difficult. This article proposes a new approach that doesn't need any sort of previous expert demonstration data. To do so, the authors, propose to incorporate the techniques [Hindsight Experience Replay \(HER\)](#) and [Curriculum Learning \(CL\)](#), hindsight experience replay, and curriculum learning, respectively, with [Reinforcement Learning \(RL\)](#), with this the robot will learn how to navigate in a dense crowd. In this approach "the goal-oriented task is formulated as a sparse reward problem". In this work, the authors developed two different models, RL+RS, [Reward Shaping \(RS\)](#), and RL+HER+CL, and after experimenting with both they came to the conclusion that the model with RL+HER+CL can learn the human-aware navigation policy and can navigate in a more efficient way, in terms of time consumption and trajectory length when compared with the "state-of-the-art methods that use additional demonstration data in training". This approach is very important, mainly for robots that will operate with companies that don't have enough money to get the datasets and otherwise would opt not to have the robot.

In [22], it is presented, as the authors call it, as a "tunable human-aware robot navigation planner" this new approach is capable of adapting to different contexts and therefore it makes the robot's navigation more efficient and more social. For this, they added Human Safety and Human Visibility costmap layers to both the global and local costmaps. These new layers are modeled as a 2D Gaussian and a 2D Half Gaussian, respectively, the Human Safety layer and the Human Visibility layer, the addition of these two layers is controlled by the Human State of the [Human-Aware Timed Elastic Band \(HATEB\)](#) local planner. HATEB is part of a previous work done by these authors that can be read in [23]. The HATEB local planner accesses different scenarios and determines the Human State and the Planning State. These two states decide the planning mode and the control of the transition between modes. The tests that were made by the authors conclude that the robot is able to behave well in situations like waiting for a human to pass on a corridor or avoiding a human going in the same path as the robot in opposite directions [fig.2.8](#). This approach makes it possible for the same robot to navigate, for example, indoors and outdoors, and be able to adapt to each environment.

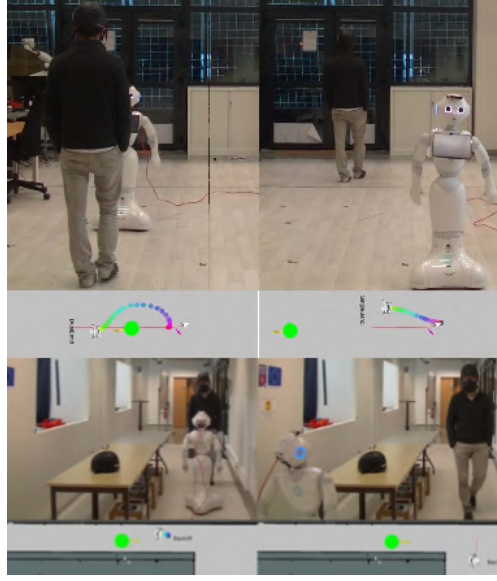


Figure 2.8: The author’s system deployed and tested on a real robot, Pepper. The top pictures show the open space scenario and the bottom pictures show the test of the Backoff-recovery mode [22].

2.4 Ultraviolet-C radiation

UV light has three wavelength categories: **Ultraviolet-A (UV-A)**, **Ultraviolet-B (UV-B)**, and **Ultraviolet-C (UV-C)**. UV-C technology, which is based on ultraviolet-C (UV-C) radiation, which is known for its great sterilization and disinfection characteristics. A high-energy type of ultraviolet light, UV-C radiation, has wavelengths between 200 and 280 nanometers. It is very useful in a variety of applications because of its capacity to kill bacteria by breaking their **Deoxyribonucleic Acid (DNA)** apart.

The early 20th century marked the start of the historical development of UV-C technology as scientists began to realize its potential. Niels Finsen won the 1903 Nobel Prize in Medicine for his use of UV light to treat lupus vulgaris [24]. One of the first inventions was the use of mercury vapour lamps that released UV-C light in medical and water disinfection situations. UV-C technology has advanced significantly over time.

UV-C technology is used in many different industries nowadays. UV-C is used in healthcare to disinfect surfaces, the air, and water, which helps with infection control. UV-C is used in the food sector to process and package food to ensure food safety. UV-C technology is used in water treatment facilities to clean wastewater and clean drinking water. UV-C is used in air filtration systems to improve indoor air quality. UV-C is used in laboratories and research facilities for experiments and studies.

The use and reach of UV-C technology have increased as a result of recent technological advancements. Longer lifespans and more energy efficiency are features of improved light designs. With the development of energy-efficient UV-C LEDs, [Light Emitting Diodes \(LED\)](#), portable and small UV-C devices are now possible. UV-C technology is incorporated into smart and IoT-based, [Internet of Things \(IoT\)](#), systems for automated control and accuracy.

Despite its benefits, UV-C technology has disadvantages. The need for proper shielding and control mechanisms is driven by safety concerns over human exposure to UV-C radiation. Another aspect to take into consideration is the compatibility of the materials, as some materials deteriorate when subjected to UV-C light. For UV-C applications to be secure and productive, adherence to regulatory standards and directives is essential. The disposal of UV-C lamps and LEDs is one environmental concern that needs to be addressed.

In conclusion, UV-C radiation-driven technology has revolutionized purification, sterilization, and disinfection in several kinds of industries. UV-C technology continues to make a substantial contribution to environmental sustainability and public health from its historical roots to present uses. UV-C technology will continue to be crucial as we work toward creating healthier and safer environments for everyone.

UV-C lamps use a formula to calculate the dosage of radiation that is emitted by the lamp, which is:

$$\text{UV Dose (in mJ/cm}^2\text{)} = \text{UV Intensity (in mW/cm}^2\text{)} \times \text{Exposure Time (in seconds)}$$

In this formula, there are three parts: the UV dose needed in mJ/cm^2 , the UV intensity in mW/cm^2 , and the exposure time in seconds. The UV dosage needed to disinfect, according to [25], "It would then be safe to assume a required UV dose higher than 20 mJ/cm^2 for likely 99.9% reduction." This way, in this dissertation, the dosage of reference will be 20 mJ/cm^2 .

SUPPORTING CONCEPTS

3.1 Robot Operating System (ROS)

With the rising number of robot companies and manufacturers, it would be impossible to program each robot and update the functionalities of a robot if each had a different interface and program language. It would be impossible to program robots from different manufacturers to work together and it would also be highly inefficient. Building and programming robots requires a lot of communication and coordination between the various components of a robot, sensors, actuators, batteries, and processors. More so, even if robots that came with all these components connected and with an interface, most likely, each company would have its own interface, and it would be very time-consuming to have to program each robot in each company's interface. The Robot Operating System, or ROS, "is a set of software libraries and tools that help you build robot applications." [26]. ROS is an open-source software that helps developers connect all the components of a robot seamlessly. With ROS developers can focus on their real goal with the robot, other than spending weeks trying to figure out how to work with the robot. ROS has a vast set of tools such as simulations, with ROS simulations it's possible to create a digital twin of a robot and test it as many times as needed before trying it on the real robot. This not only prevents accidents but also allows different developers to test on "the same robot" rather than have to wait for the robot to be free so they can test their functionalities.

In May 2021, ROS released ROS2, ROS1 was originally released in 2007 and since then a lot has changed in robotics. ROS2 release was to compensate for those changes, with ROS1 being the "de facto global standard for robotics" [27] ROS2 was built to be safer, more reliable, and easier to use. ROS2 was built with the requirements of industry-grade companies. Like ROS1, ROS2 is open-source free software, made to be used by everyone from consumer to industry level, in every situation.

To make an operating system as complex and organized as ROS or ROS2 there needs to be some concepts that are well structured and have a clear function. In ROS2 these concepts make up the "ROS (2) graph", the following subsections will go through and succinctly explain concepts such as Nodes, Topics, Services, Parameters, and Actions.

3.1.1 Nodes

Nodes are the basis of ROS, each node should do a simple task, such as publishing the information of a sensor. A node can send and receive data from other nodes via topics, services, actions, or parameters. In ROS2 an executable file can have one or more nodes.[28] In the figure bellow, fig.3.1 there are three nodes, one is a publisher and a service server, one is just a subscriber and the other one is a subscriber and a service client, in fig.3.1(a), the publisher sends a message that, in fig.3.1(b), is delivered to the two subscribers.

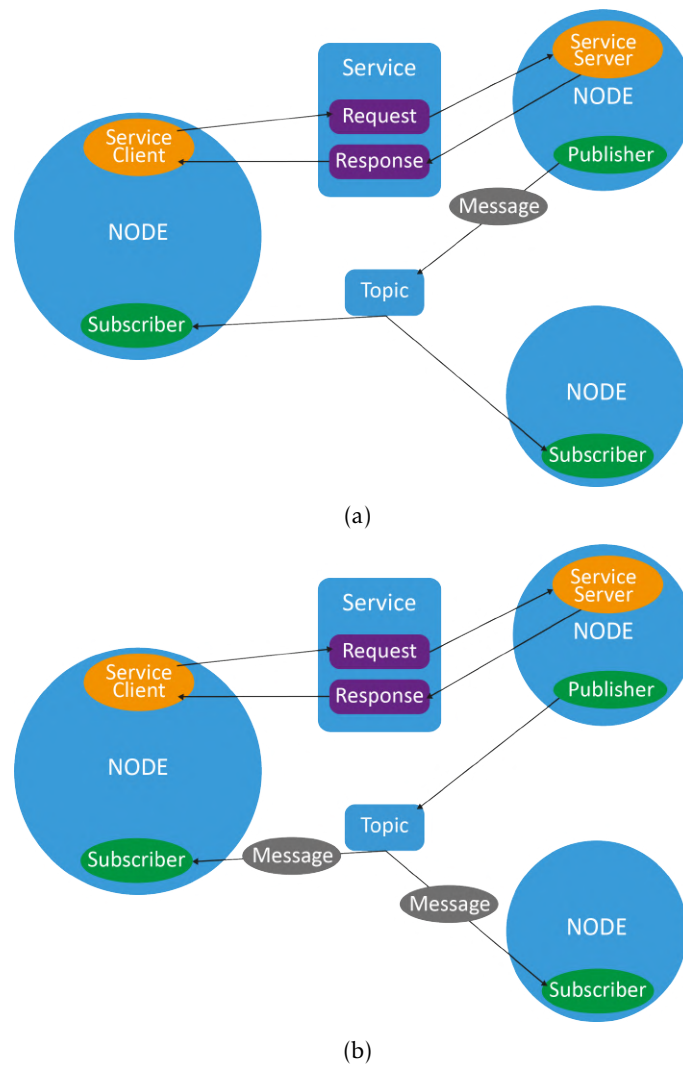


Figure 3.1: ROS node example, based on [28]

3.1.2 Topics

Topics are used by nodes to send or receive data, nodes can send information to any number of topics and can also receive information from any number of topics, which act as a bus for nodes[29]. In the figure below, fig.3.2, there are four nodes, two publishers, and two subscribers. As shown in the sequence, first, one of the publishers sends a message through the topic and the two subscribers receive it, after that the other node also sends a message through the same node and the two subscribers receive it.

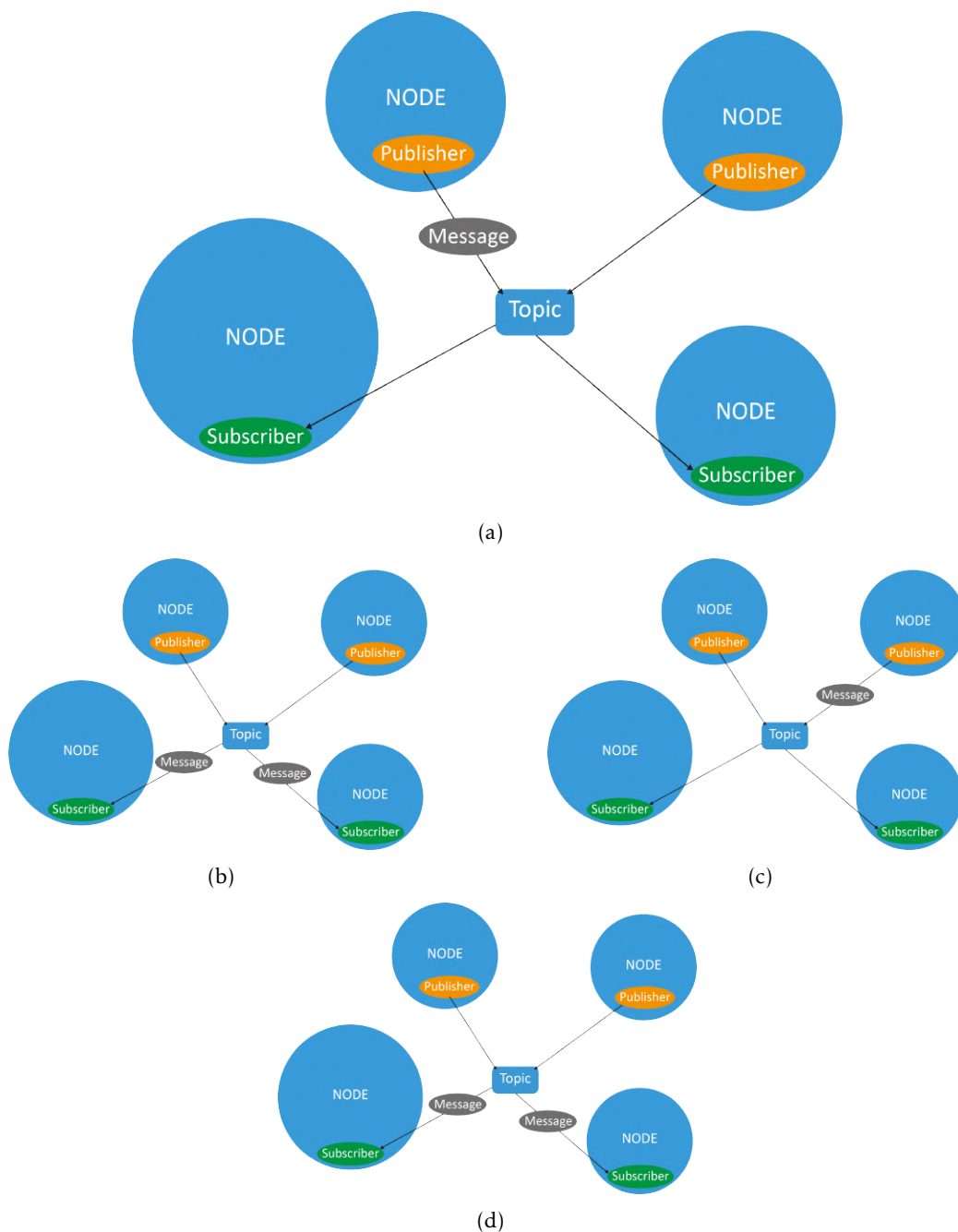


Figure 3.2: ROS topic example, based on [29]

3.1.3 Services

Similarly to topics, services are used to transfer data between nodes, the difference between services and topics is that services have a request-response base so a node sends a request, and another node responds but the node responding will only do so when asked. In topics, nodes keep receiving information as long as the other node sends it [30]. The same service can be used by different nodes but only one node can use it at a time. In the figure below, fig.3.3, it can be seen an example of a node requesting a service and another node responding to the request.

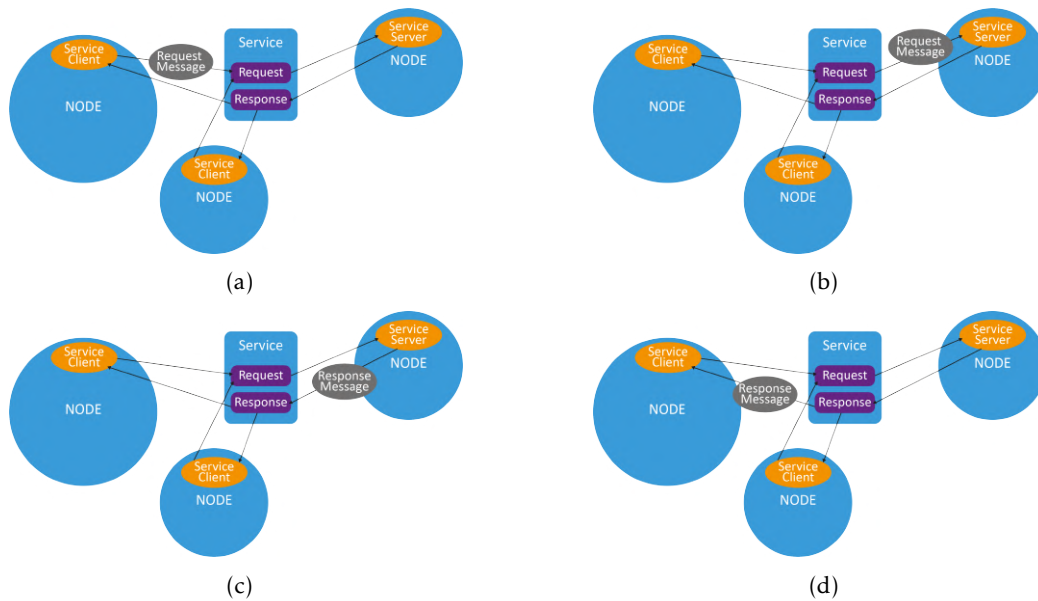


Figure 3.3: ROS service example, based on [30]

3.1.4 Parameters

Parameters are something like node settings. Parameters are used to configure nodes. Each node has its own parameters and can use parameters to receive data.[31]

3.1.5 Actions

Actions consist of three parts: a goal, feedback, and a result. Actions are built on topics and services. They work like services but may be canceled and provide continuous feedback, while services only return a single response.

In actions, as can be seen in the following figures, the “action client” node sends a goal request to an “action server” node, through a service, fig.3.4, the "action server" responds letting the “action client” node know that it got the goal, fig.3.5, then the "action client" sends a result request through another service and waits for the response, fig.3.6, while waiting for the response the "action client" will receive continuous feedback from the "action server", fig.3.7, when the goal is achieved the "action server" will send a response through the service used before, fig.3.8 [32].

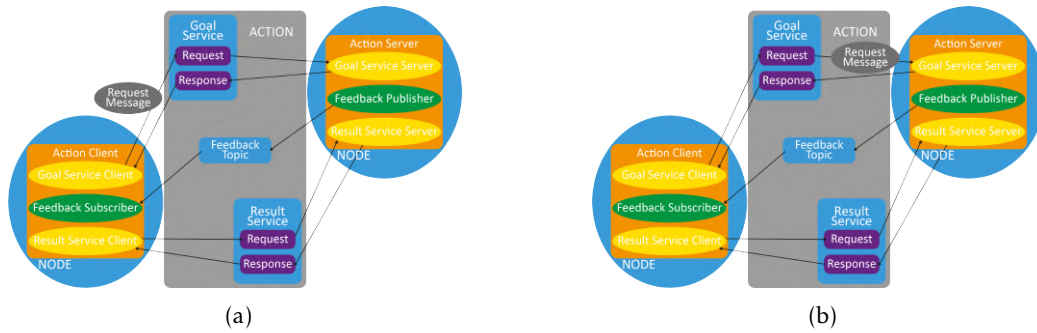


Figure 3.4: ROS action example, goal request, based on [32]

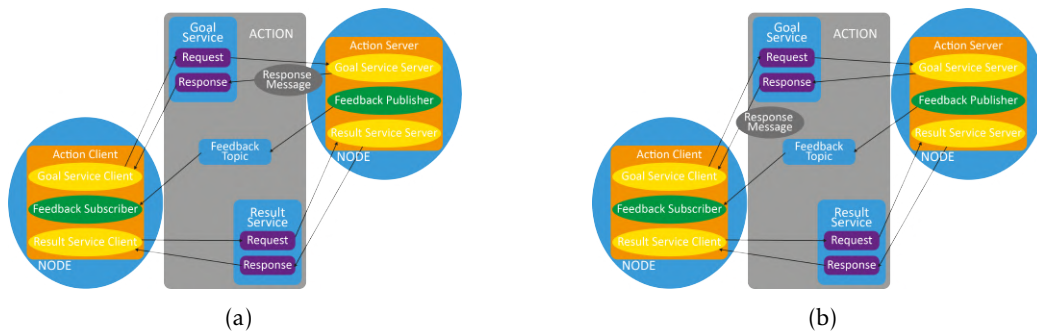


Figure 3.5: ROS action example, goal response, based on [32]



Figure 3.6: ROS action example, result request, based on [32]

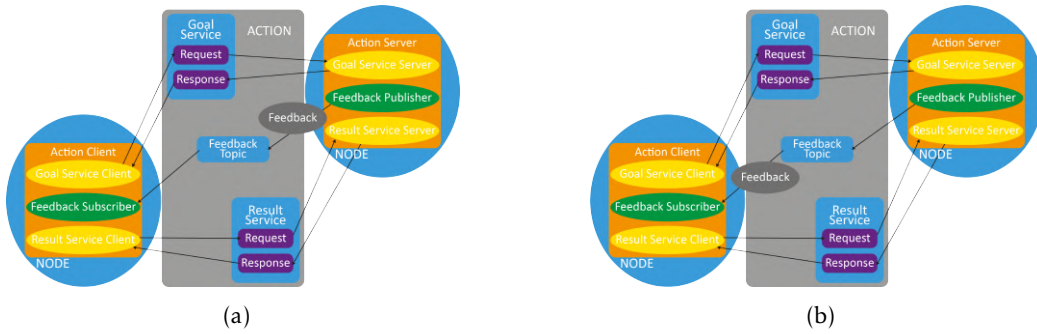


Figure 3.7: ROS action example, feedback, based on [32]

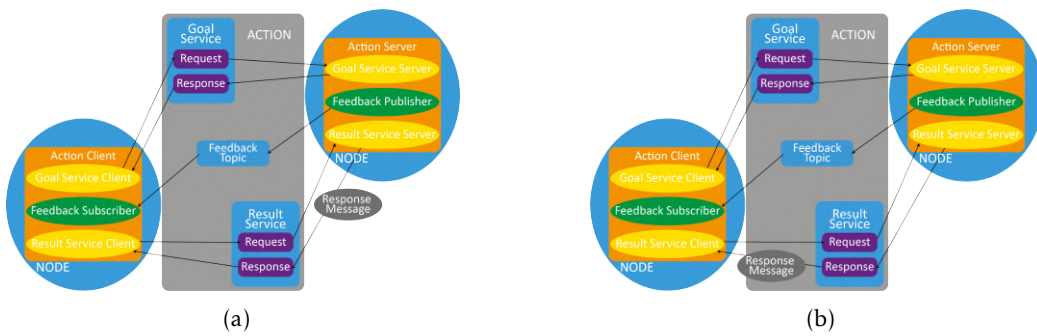


Figure 3.8: ROS action example, result response, based on [32]

3.2 SLAM

SLAM, or Simultaneous Localization and Mapping, is “the process by which a mobile robot can build a map of the environment and, at the same time, use this map to compute its location” [33]. This process is very important in autonomous mobile robots in order for them to move in a logical way without getting lost in an environment. At the same time, SLAM is very useful to make maps of the environments in order for the robots to use them to navigate through the environment.

Let’s look at a hospital, every hospital has a floor plan, but in that floor plan usually there aren’t beds, chairs, tables, etc. And sometimes there are small changes that are made and that are not updated on the floor plan. This is why giving the robot the ability to map the environment with the objects as they are and with the distances being measured by the same sensor that is going to guide the robot through the map with navigating is essential.

There are multiple SLAM techniques, such as visual SLAM, which use cameras for mapping and localization, and LIDAR-based SLAM, [Light Detection and Ranging \(LIDAR\)](#), which makes use of laser sensors for mapping and estimating robot positions. Each strategy has advantages and disadvantages of its own, but they are sometimes combined to achieve better results. In figure 3.9 there is an example of a robot using SLAM Toolbox.

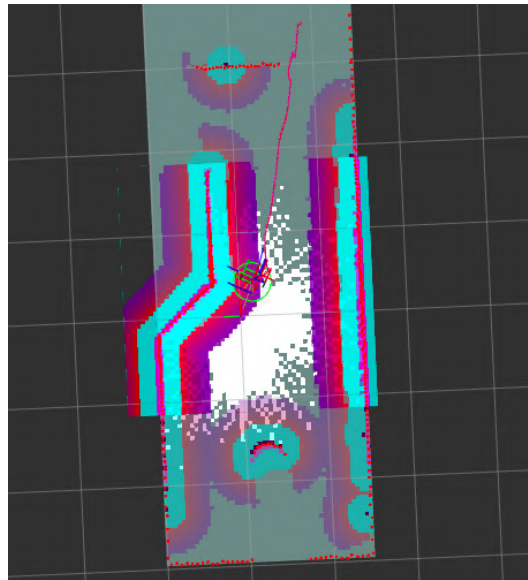


Figure 3.9: Robot using SLAM Toolbox to map an environment

PROPOSED MODEL

In a hospital, there are thousands of people going in and out every day, and many of them have some kind of disease. There are different types of transmission of germs, one of them being through contact with the skin and inhalation. Therefore, the corridors and rooms of a hospital become a place facilitative to the transmission of germs. This makes the disinfection of a hospital essential for the well-being of the patients. In this chapter, it is going to be discussed the proposed model to help with this disinfection. As it can be seen in figure 4.1 the proposed model has four major blocks, the map, in this case of a hospital, or part of it, the planner, to choose the waypoints, and order them, in a way that the robot disinfects every part of the hospital, the navigation, which is going to be used for the robot to follow the waypoints chosen by the planner and at last the platform which is the robot, its sensors, and lights.

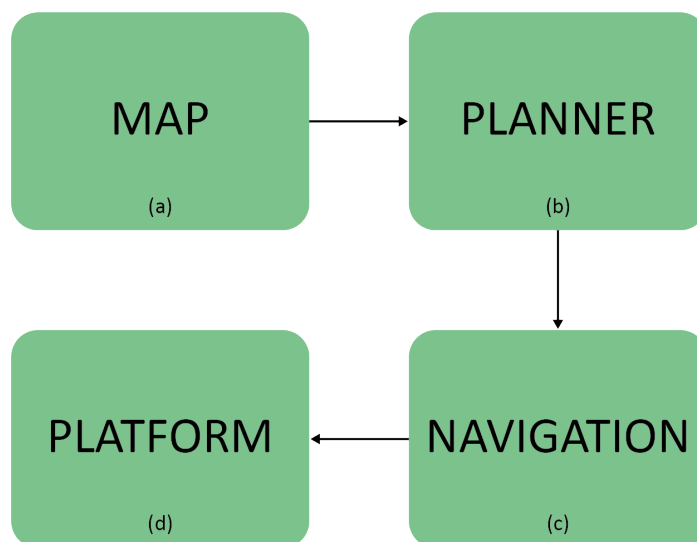


Figure 4.1: Proposed Model

4.1 Map

With mobile autonomous robots navigation is essential, particularly in hospitals where efficient movement and precise navigation can save significant time disinfecting and avoid disturbing the people in the hospital. This section focuses on the importance of having a precise map for the robot to have a base. In this particular case, having an updated map is essential, not only because the planner needs to have a map as close to reality as possible to plan the waypoints in the correct places, but also because it is a hospital where different things can be moved around and be in different places. To explain the different steps of the system it will be used the map illustrated below, fig.4.2, as an example.

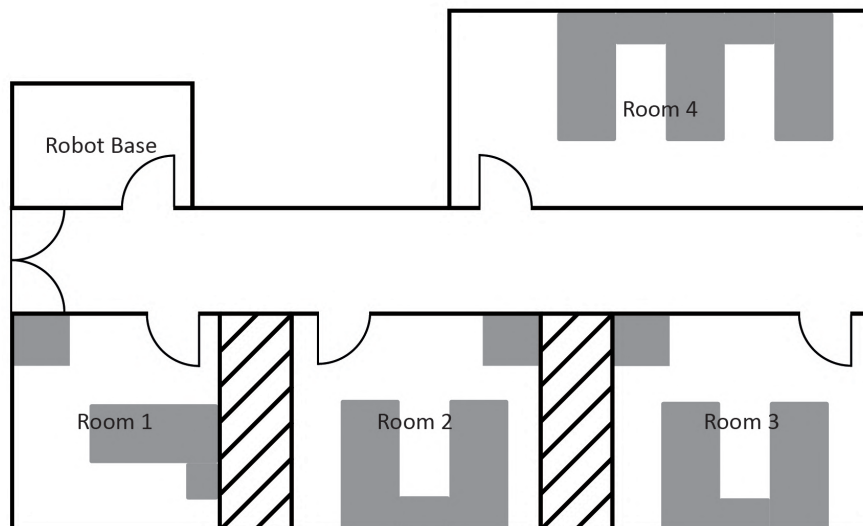


Figure 4.2: Hospital floor plan

Let's take a closer look at a single hospital room, in a room there is a bed, an IV stand, a patient support table, and a chair for any visitor since disinfection has to be made when there is no one in the room, the IV stand can be removed from the equation, as it usually is only in the room when there is a patient. The bed usually doesn't move from patient to patient because it has a specific place to be in the room in order to be connected to the power and near some outputs such as oxygen, for the table and the chair, it is normal that they are moved from one patient to the other. For example, one patient can have a long visit that wants to be near and puts the chair near the bed making the layout of the room as in fig.4.3(a), the next patient might not have any visitors and so the table is near the bed and the layout of the room is as in fig.4.3(b), therefore the map has to be regularly updated for the robot to have the correct layout when the planner chooses the waypoints.



Figure 4.3: Two different layouts of hospital room 1 - Render

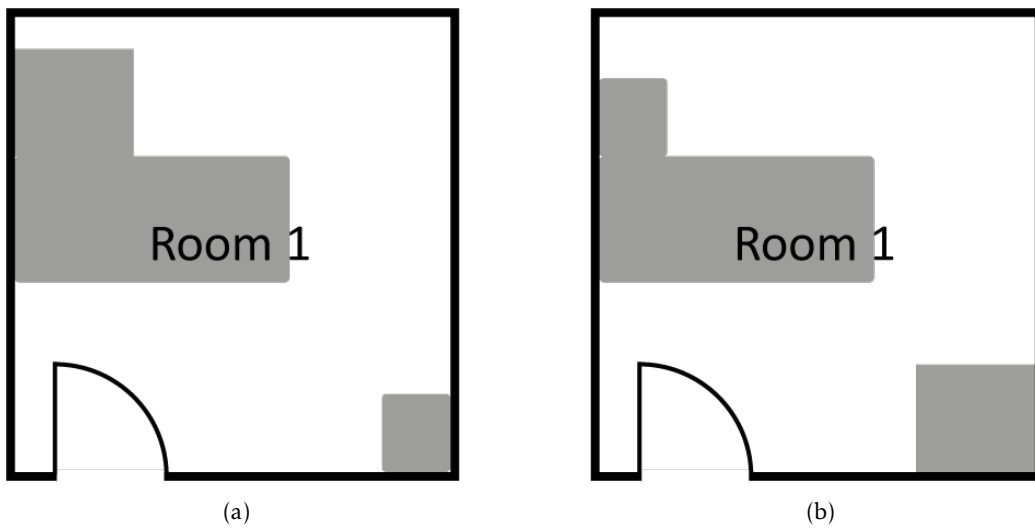


Figure 4.4: Two different layouts of hospital room 1 - Map

4.2 Planner

After loading the map and the parameters adjacent to it, such as the map origin and the resolution, the software should determine points of interest for the robot to go to. To determine these POIs, [Point of Interest \(POI\)](#), it needs to consider the radius of the light beam emitted to disinfect the area, determine the distance the robot can be from the walls and the objects in the rooms, in order to disinfect the room and the objects. A small scheme of the light and the light beam is illustrated in fig.4.5. The radius of a light beam is not a specific distance like the size of the robot but depending on the light used, it is possible to determine how far the light is effective and for how long it has to stay on. Also, the software must know the size of the robot being used and if it has any restrictions in movement, so the robot doesn't bump into objects and the planner doesn't choose waypoints that are unreachable by the robot.

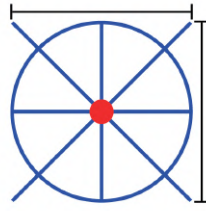


Figure 4.5: Disinfection lights radius

With the map loaded and all the necessary distances set, the planner will then start to determine the waypoints that the robot should reach. To determine these waypoints the planner needs to be sure that the chosen waypoints cover the whole area to be disinfected. As seen in fig.4.6 in the corridor, the planner had to set two waypoints, one next to the other in order for the robot to disinfect the whole corridor. The planner also needs to be sure that the waypoints chosen are not on top of some object or some wall.

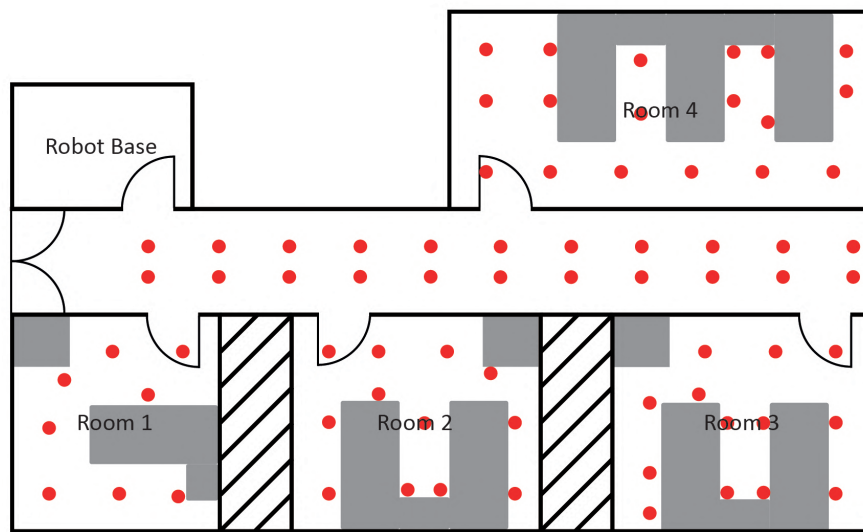


Figure 4.6: Hospital floor plan with POIs

Now with all the waypoints set, the software should order them in a way that it makes a path for the robot, fig.4.7, this path should be sequential from the starting point of the robot to the furthest point on that floor and should try to make the shortest path possible in order to be as fast as possible and make that floor or those rooms available as soon as possible. As it was said before, in hospitals there are lots of emergency's, so having a path that is short and having a robot that disinfects in a fast, but effective way is crucial.

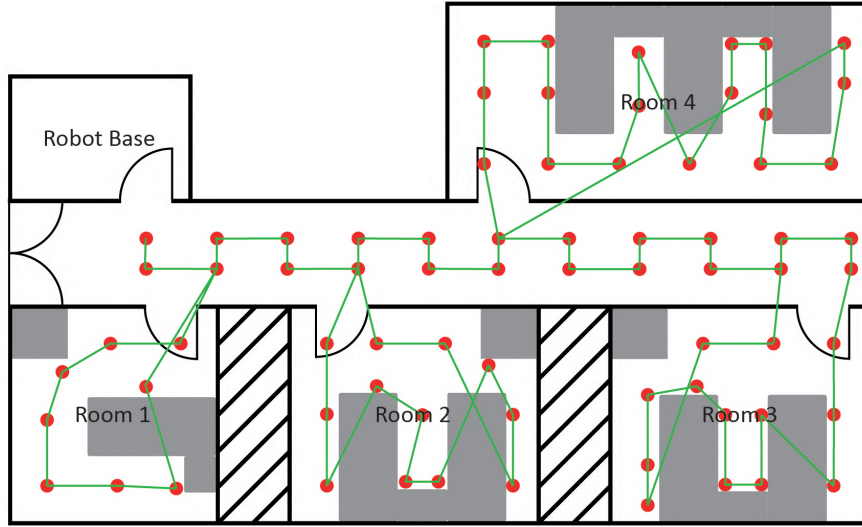


Figure 4.7: Hospital floor plan with POIs and path between them

At each POI the robot should stop, make sure that there are no humans in the range of the lights, and emit a sound to let potential people going in the room know the robot is disinfecting. It should turn the disinfection lights on and keep them on for a certain period of time depending on the lights, the intensity of the lights, and the area it covers.

With all the processes described, it is also necessary to have some requirements regarding the implementation environment and the robot's use. As was said before this robot will be used in a hospital environment, which means that there are a lot of people attending the facilities and that those people are not controlled or traced. In this kind of environment, there is usually a wifi network, one that is public most of the time and doesn't have any kind of password. The robot needs a network connection to be able to move and that connection cannot be by cable, as the robot ought to move freely through the hospital. Therefore, this whole system must have strong cybersecurity measures, and there should be a private and secure network only for the robot. All robots can be dangerous if hacked, but in this particular case, the robot is in a hospital where people are inherently more fragile and thus, easily hurt, and the robot will be equipped with strong disinfection UV-C lights that can blind people if misused.

Since these robots are expensive and assuming that the robot won't be needed to disinfect every time, as long as its battery allows it, the planner should have some kind of structure to make the system easy to add new functionalities. This would allow the hospital robot to transport utensils around the hospital, to help doctors and nurses at their jobs, and even to help some people get around the hospital. This structure should also have actions like going back to the base to charge, opening doors, pressing buttons, and asking for help if stuck somewhere. This planner has to be flexible in a way that can be used in different environments, such as factories, supermarkets, train stations, and even restaurants.

As discussed in the map section, hospital rooms tend to have some of the objects moved from one patient to another, or from one disinfection to the next. In fig.4.3, where in fig.4.3(a) the chair is by the bed and the table is in a corner of the room, in this case, the planner should set waypoints and it should order them as shown in fig.4.8(a). But if the room layout is like in fig.4.3(b), the waypoints and the path will be like fig.4.8(b). The planner has to have the ability to change the waypoints chosen and the order of those as the layout of the rooms changes. If for example, the last patient leaves the room with the layout shown in fig.4.9, the robot will not be able to disinfect the whole room because the path is blocked by the chair and the robot will not have the ability to move objects, also because that could be dangerous, as the robot would not know if there was anything fragile on the chair or the patients' table.

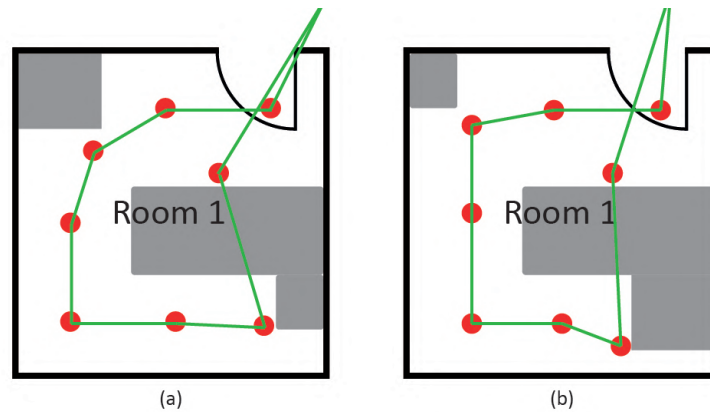


Figure 4.8: Hospital rooms with different layouts and paths

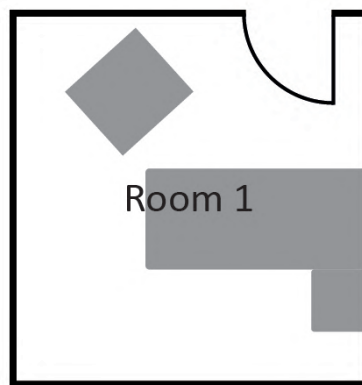


Figure 4.9: Hospital room with different layout and path blocked

4.3 Navigation

Navigating is "the act of directing a ship, aircraft, etc. from one place to another, or the science of finding a way from one place to another"[34], every human needs some kind of navigation knowledge in order to move from one place to another, to drive a car from one place to another, etc. And like humans, mobile robots also need to know how to navigate in order to move from one place to another, if the robot is remotely controlled, that navigation will be made by the human controlling, it but if it is an autonomous robot the navigation will need to be made by some kind of software that can plan a path from one point to another. In this specific case, as illustrated in figure 4.10, the software must be able to receive a list of waypoints to follow and must be able to follow the order of the list. It has to be able to plan different paths from one waypoint to another in case the shortest route is somehow obstructed and the robot can't pass. This navigation software needs to be secure, just like the planner it needs to have some kind of cybersecurity to make it harder to hack and control the robot in an unwanted way.

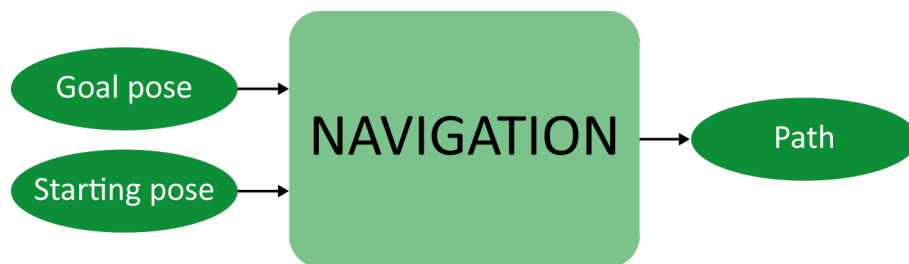


Figure 4.10: Navigation

4.4 Platform

At last the platform, in order to make all these systems come to life and reflect on some physical action is needed a mobile robot. Taking into consideration the environment, a hospital, the robot cannot be too big, in a way that it can fit in doors and some small corridors between beds and objects, it can't be too small either because it needs, not only, to be able to carry the disinfection lights and the sensor, but also to be some kind of sturdy as it will have a lot of use and wear. The robot has to be able to receive the navigation system discussed before, it has to have sensors in order not to bump into objects that can be moved and that are not on the map, and sensors to sense if there is any person in its range to start disinfecting in safety. It should also have UV-C Lights to disinfect and a speaker to emit some sounds. The robot should also have a lot of flexibility and agility to make it possible to make tight turns and even turn around in small spaces. It needs to have a battery large enough to travel and disinfect an entire hospital floor without having to go back to charge, in order to make the process as efficient as possible. It needs to have good suspension and sturdy wheels to be able to climb small bumps or even cables lying on the floor. It doesn't need to have high-speed capability, but it should move at more than 4km/h in order to reduce the time moving from one place to the other. In figure 4.11 there are some examples of mobile robots that could be used.



(a) [35]



(b) [36]



(c) [37]

Figure 4.11: Autonomous Mobile Robot examples

IMPLEMENTATION

This chapter provides an overview of the implementation of the proposed model discussed before. For each block it will give a detailed explanation, for each block, on the decisions made and the software and hardware chosen for each function. Furthermore, it explains the function of each software and hardware and its advantages and disadvantages in this use. Finally, it will explain in detail the planner created and the connections between the different blocks. Fig.5.1, shown in the previous block scheme, provides a visual to how the system works.

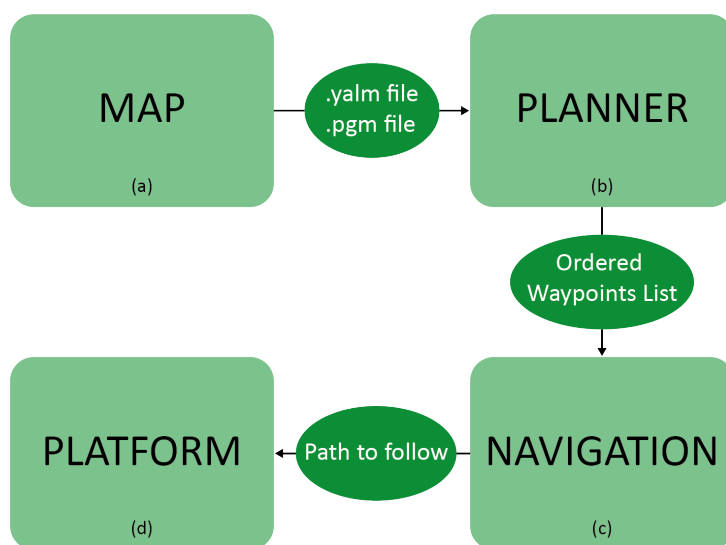


Figure 5.1: Implementation Block scheme

5.1 Platform

This chapter begins with the platform because that was the starting point of this implementation, although this dissertation is meant to be as abstract as possible, in a way that it can be applied to any robot, there was a model already in mind to apply this dissertation and to simulate its ideas.

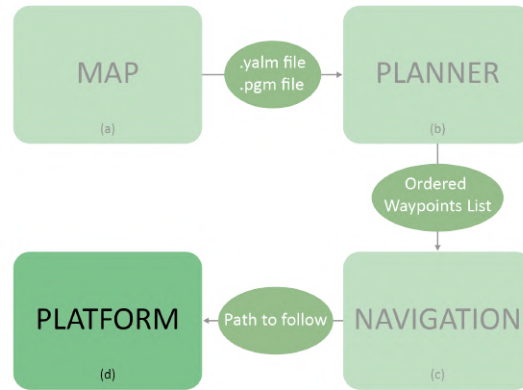


Figure 5.2: Implementation Block scheme - Platform

The robot that will be used is AGILEX TRACER, fig.5.3. TRACER is a two-wheel differential **Automated Guided Vehicle (AGV)** that, because of its size and flexibility, is agile and has a lot of potential in indoor situations [35]. Being a differential robot means that the robot has two separately driven wheels that are situated on each side of the robot body. With this the robot cannot move sideways, making the maneuvers harder. With this being said, differential robots are widely used because they are easier to program and control and they are much cheaper than an omnidrive robot.



Figure 5.3: Tracer by Agilex [35]

TRACER has a length of 685mm, a width of 570mm, and a height of 155mm without any extra equipment. It weighs 28kg, has a minimum ground clearance of about 30mm, and handles a payload of 100kg. The maximum speed is 6km/h, the batteries handle 4h of operating time and take 2h to charge, and the minimum turning radius is 0m as it can rotate in place [5.4](#).

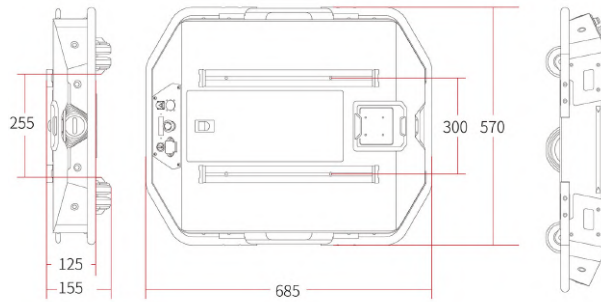


Figure 5.4: Tracer by Agilex - Top view [\[35\]](#)

For the sensor, the robot will be equipped with a Hokuyo UTM-30LX Scanning Laser Rangefinder, [fig.5.5](#), which has a long detection range of 0.1 to 30m, a wide angle of 270°, it is made for an outdoor environment and is compact and light with 60mm of width and height and 87mm of depth, weighing only 370g. This sensor needs a power source of 12VDC, it has an accuracy of 30mm from 0.1m to 10m and 50mm from 10m to 30m, has an angular resolution of 0.25° and scans at 25msec/scan, the sound level is less than 25dB, the interface is USB2.0 and it works in temperatures going from -10°C to 50°C, with less than 85% humidity without dew or frost [\[38\]](#).



Figure 5.5: Hokuyo UTM-30LX Scanning Laser Rangefinder [\[38\]](#)

For the disinfection lights, the lights chosen for this case were, TUV Amalgam XPT from Philips, [fig.5.6](#). This lamp has a cap-base G10.2Q, operates in a universal position, and its main use is for disinfection. It has 180W of power, 2.1A, and needs 90V for ignition. The lamp is dimmable and has a depreciation of 15% [\[39\]](#).



Figure 5.6: Philips TUV Amalgam XPT [39]

Using the formula explained in the State of the Art chapter, having the lights and the instruments to measure the light intensity it would be possible to calculate the time needed for the light to be on in order to achieve $20 \text{ mJ}/\text{cm}^2$. These instruments are needed because the area covered by the light of the lamp depends on various factors such as the size of the lamp, the shape of the lamp, the angle the lamp is positioned, etc.

This robot would need an operating system in order to connect and use the information of the sensor and have the rest of the system running. In this case, the operating system chosen to use in this robot was ROS2. ROS has been the leading operating system in robotics for quite some time, so using the latest version would be the best choice. ROS2 is not only the latest version of ROS but also brings new features with the new update; in this version, more than one robot can be the master, which means that if one of the robots gets lost or disconnected from the network, the others can continue their work. This is unlike ROS(1) where there was only a master and so if the master lost connection to the network, all the other robots would stop working. Also, in this version, there is a new layer of cybersecurity which, as explained in the previous chapter, is crucial for the project's success in this particular environment.

At the time of this dissertation, the robot is not ready for testing. The robot is still being assembled and its current state is shown in the figures below, fig.5.7.



(a)



(b)

Figure 5.7: Robot under construction

5.2 Navigation

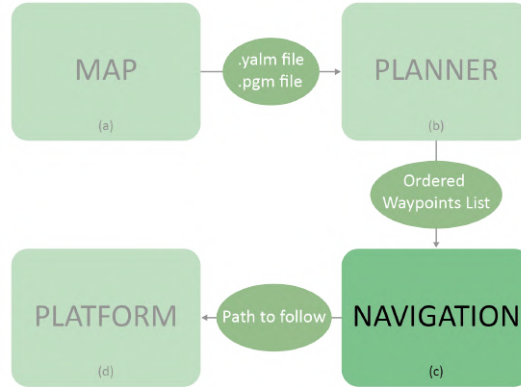


Figure 5.8: Implementation Block scheme - Navigation

Having chosen ROS2 as the operating system was a starting point for choosing which navigation to use. It had to be a navigation system that could send orders to the robot in ROS2 to make it move around, but it also had to be a navigation system that could receive a list of waypoints from the planner and follow them in the order set by the planner. For this reason, the navigation system chosen for this dissertation was Nav2, Nav2 "is the professionally supported spiritual successor of the ROS Navigation Stack"[40]. The goal of this project is to be able to control a robot in a safe way in order for it to complete complex tasks, this system not only has the features to orient a robot to go from point A to point B but also it can have intermediary poses and it can also do other tasks like object following. Nav2 is a framework used in more than 50 companies around the world, it is a high-quality framework. Having a framework like Nav2 is essential for this type of project, Nav2 "provides perception, planning, control, localization, visualization, and much more to build highly reliable autonomous systems."[40]. Nav2 uses Behaviour Trees in order to create more efficient and intelligent navigation by coordinating a number of separate modular servers. These servers, communicate with the Behaviour trees through ROS interface. A robot can use different Behaviour Trees to perform different tasks. In fig.5.9. There is a diagram showing the structure of Nav2.

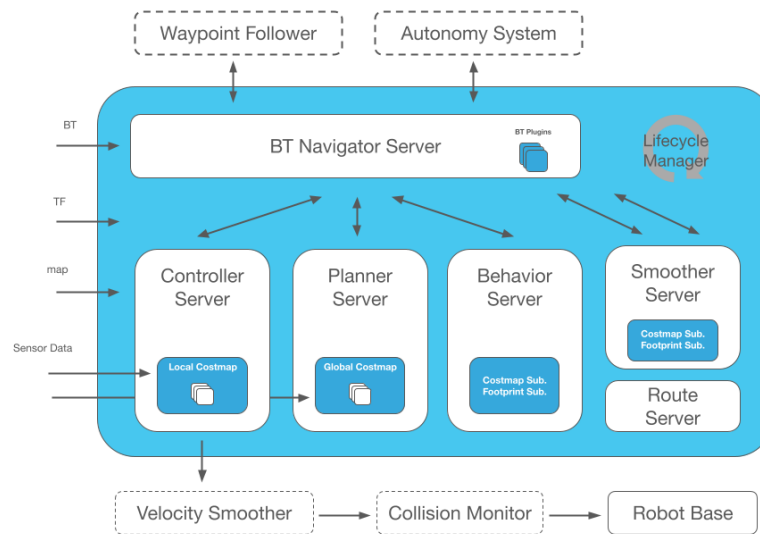


Figure 5.9: Nav2 architecture [40]

Nav2 has tools like Simple Commander which is "Python3 [Application Programming Interface \(API\)](#) to interact with Nav2 in a pythonic manner"[40], Map Server to load, save, and store maps, [Adaptive Monte Carlo Localization \(AMCL\)](#) to locate the robot on the map, and Nav2 Planner to plan a route from one point to another avoiding obstacles. It also has Nav2 Controller to control the robot while following a route, Nav2 Smoother used to smoothen the path in order to make it feasible, and Nav2 Costmap 2D, to use the information from the sensors to make a cost map of the world. Moreover, it is equipped with Nav2 Behaviour Trees and [Behaviour Trees \(BT\)](#) Navigation to create complex robot behaviors, Nav2 Recoveries to recover the navigation in case of a failure, Nav2 Waypoint Follower to make the robot follow sequential waypoints. Finally, it also has Nav2 Lifecycle Manager to manage the lifecycles and monitor the servers, Nav2 Core to enable new algorithms and behaviors, Collision Monitor to monitor sensor raw information in order to detect any collision, and Velocity Smoother, to smoothen output velocities to make sure the commands are feasible [40].

5.3 Planner

With the robot chosen and the navigation framework being the Nav2, the planner is the next block, this block is going to send a list of ordered waypoints to the navigation block, in order for the robot to follow them and disinfect the whole hospital floor, or parts of it. This planner will be responsible for analyzing a map, previously made, setting waypoints on the map, and ordering them in a way that is efficient and covers the whole area to disinfect. For the explanation of this planner, the turtlebot example will be used to show some steps of the planner in question.

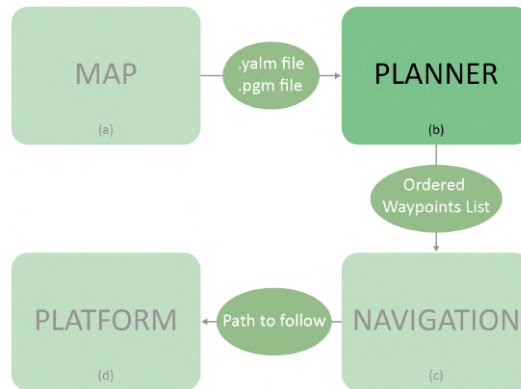


Figure 5.10: Implementation Block scheme - Planner

The planner starts by calling the "DesinfectionService()" which has a callback where the function "points_in_map()" is called. In this function, there are two cycles, one inside the other. The first one is used to go through the x-axis. The For cycle is going to increment the value of the x variable by the distance wanted between two waypoints, set prior, the second cycle will do the same thing but for the y-axis. Inside the second cycle, it is verified if the waypoint, given by the coordinates of the cycles, has the color that, on the map, symbolizes the obstacles found by the robot. If the coordinates are not of a pixel coincident with an obstacle, those coordinates are added to the waypoint list. Still inside the if, the function "in_radius()" is called. This function verifies if the waypoint added to the list is far away enough from the obstacle so that the robot doesn't hit the obstacles, because of its size. After this part of the code, there is already a list of waypoints distributed in the map, as shown in fig. 5.11, in this figure the waypoints are represented by red pixels, the wall of the environment and the obstacles are represented in black, the free space inside the room is represented by white and unknown represented in light grey.

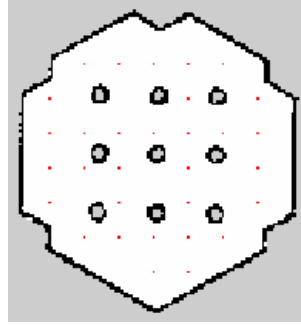


Figure 5.11: Turtlebot3 Simulation. Representation of the waypoints distribution. Waypoints in red, walls of the room and obstacles in black, free space in white, and unknown space in light grey

Going back to the callback, after the function `"points_in_map()"`, the function `"order_points()"` is called. This function orders the waypoints so that the robot goes from one room to the other in a logical, and efficient order. To do so for each waypoint on the waypoint list the function `"get_neighbors()"` is called and the waypoint in those neighbors is the next waypoint to follow. After this function `"order_points"`, there is a list of the waypoints, yet this time they're ordered from one point of the map to the one farthest from it. This allows the waypoints to be connected by order and shows the path that the robot would take if it went in a straight line from one waypoint to another, as illustrated in fig.5.12.

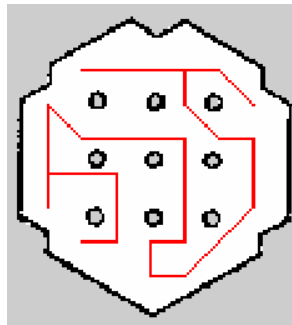


Figure 5.12: Turtlebot3 Simulation. Representation of the path made by the planner. Path in red, walls of the room and obstacles in black, free space in white, and unknown space in light grey

After that, all the waypoint coordinates are converted from pixel coordinates into spacial coordinates in the map. Then, the function `"followWaypoints()"`, from BasicNavigation from Nav2, is called and it starts by formatting the information of the waypoints into a pose format, with orientation, position, stamp, and frame id. Then this list of poses is sent to the navigation block where Nav2 will calculate the path from one waypoint to another, as it can be seen in fig.5.13 from one waypoint to another the robot doesn't always go in a straight line, this is because Nav2 calculates the most efficient route from one waypoint to another rather than just tell the robot to go in a straight line.

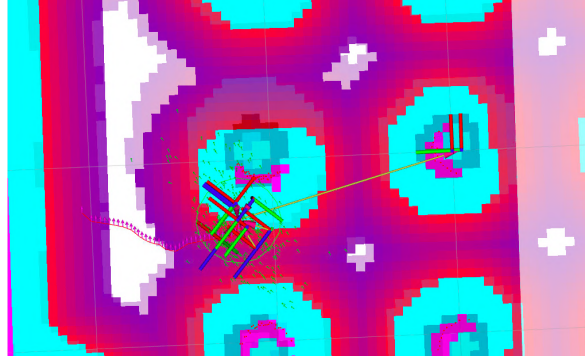


Figure 5.13: Turtlebot3 Simulation. Representation of a path between two waypoints planned by the Nav2 navigation. Walls are represented in light purple, free space is represented in white, light blue is space where the robot cannot go because of its size, and a gradient of colors, from dark purple to dark red, identifies from the free zone to the forbidden zone respectively. There are also the blue, green, and red lines that identify the axis from the robot, the green thin line that is the distance to the TF, and the pink path with a small arrow is the path created by the Nav2 navigation.

5.4 Map

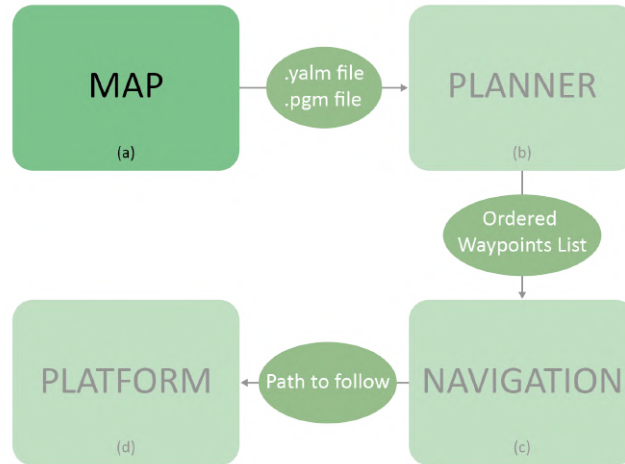


Figure 5.14: Implementation Block scheme - Map

As mentioned in the previous chapter, for the planner to choose the waypoints and their order, it needs a map. Although hospitals have floor plants, when programming robots, the best way to have a map of the floor with the objects in their places is to make one using the robot's sensors. In this case, the planner receives a .yaml file that then calls a .pgm file. To have a map in those formats with accurate information that allows the robot to navigate and get to the waypoints needed, it was used the Slam Toolbox [41]. The Slam Toolbox is a set of tools that are capable of doing 2D SLAM, SLAM as explained previously in this dissertation, stands for Simultaneous Localization and Mapping. Slam

Toolbox allows users to make ordinary point-and-shoot 2D SLAM with mobile robots, it can refine, remap, or continue mapping a saved pose-graph. It has the ability to map synchronous and asynchronous, it can merge kinematic maps, it has RVIZ plugins for interaction with the tools, and much more [41]. With this tool, the map is saved as a .pgm and also as a .yalm in order to then be loaded into the planner block. As can be seen in fig.5.15, the robot moves through space to create a map. The map is created with the information sent by the sensors and in the end, the saved map looks like the one in fig.5.16.

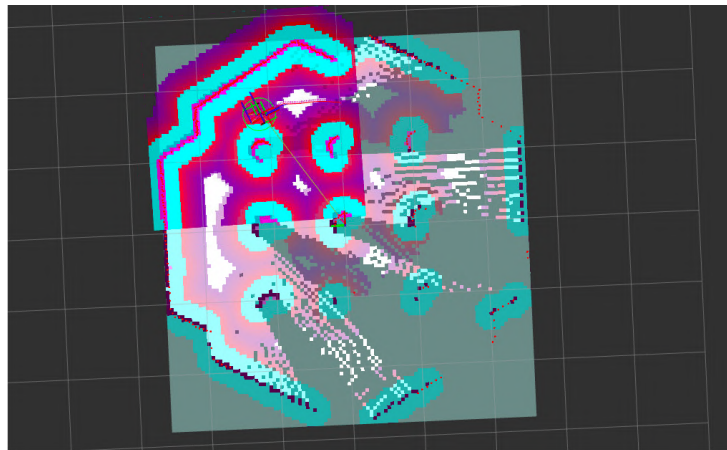


Figure 5.15: Turtlebot3 simulation. Representation of the robot mapping using SLAM Toolbox. The color code was explained before, but adding to that, there is a more intense colored square that is what the LiDAR sensor is detecting at the moment, the lighter colors are part of the map that is already created, and the green square is the size of the map at this given time.

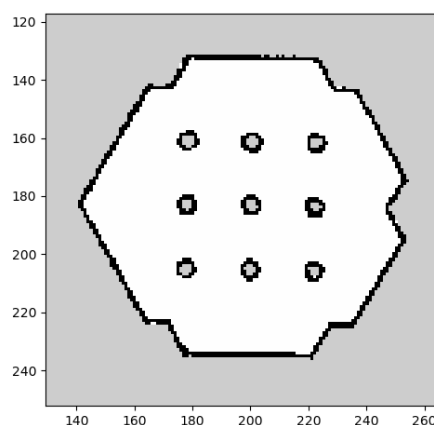


Figure 5.16: Turtlebot3 simulation. Representation of the Map created by SLAM Toolbox. The walls of the room and obstacles are represented in black, free space in white, and unknown space in light grey. The numbers on the axis are the pixel scale.

EXPERIMENTAL RESULTS

With the implementation described and explained, to test everything that was described and to draw conclusions about the final result it is needed to perform some experiments. In this chapter it will be described the experiments made, what were the different steps during the experiment, what were the results, and what could have been better in each experiment.

6.1 Turtlebot3

The first experiment was made with the tutorial of Nav2. To do so, Nav2 was installed, and after that, the Turtlebot3 packages were also installed. Everything was tested to make sure it was running correctly, and then the ROS2 service with the planner explained before, with the help of a client node, was requested.

As explained in the implementation chapter, the planner printed the map with the route for the robot to follow, [fig.6.1](#), and after that, the robot started to follow the path. At the beginning of the planner there are some parameters that need to be set up depending on the environment, such as the distance a waypoint should be from the obstacles and walls and the distance from each waypoint to another waypoint. These parameters can be determined by the strength of the UV Lights used to disinfect, by the size of the robot, and by the number of objects present in the room. If the room has too many objects then it will need more waypoints in order to reach every surface, and if the lights are not too strong it will also need to be closer to the obstacles and to another waypoint.

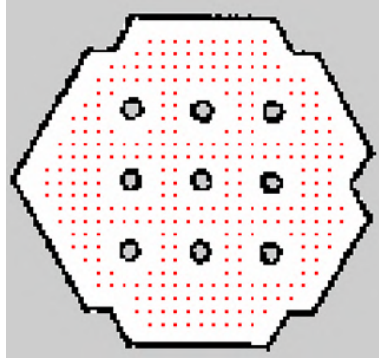


Figure 6.3: Turtlebot3 simulation. Map with waypoints, parameters 0.2m distance between waypoints and 0.2m distance to the walls. The color code is the same as the previous map created by SLAM Toolbox, with the waypoints marked in red.

On the other hand, if the lights are stronger and have a range of about 2 meters, using the parameters of 2m distance between waypoints and 0.1m distance to the walls, assuming a smaller robot, the planner choose 5 waypoints, as seen in figure 6.4.

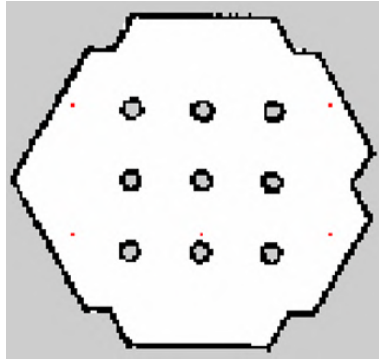


Figure 6.4: Turtlebot3 simulation. Map with waypoints, parameters 2m distance between waypoints and 0.1m distance to the walls. The color code is the same as the previous map created by SLAM Toolbox, with the waypoints marked in red.

In the figure below, fig.6.5, it can be seen that the robot correctly follows the waypoint list in the order sent by the planner. When it reaches the end, the system prints that all goals have been achieved and stops the robot. When merging the first map with the waypoints reached by the robot, it can be seen that the robot got to all the waypoints set by the planner in the map, fig.6.6.

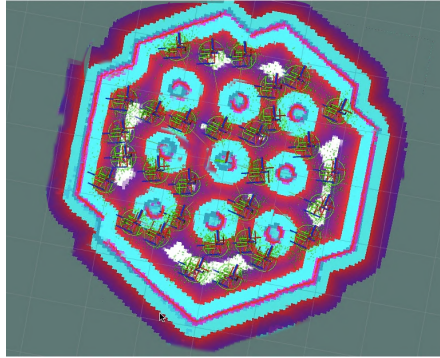


Figure 6.5: Turtlebot3 simulation. Map with all the waypoints reached. The color code is the same as the figure before that had the information from the LiDAR sensor. In this case, there are the multiple positions of the robot represented by the green circle and the different color lines representing the axis.

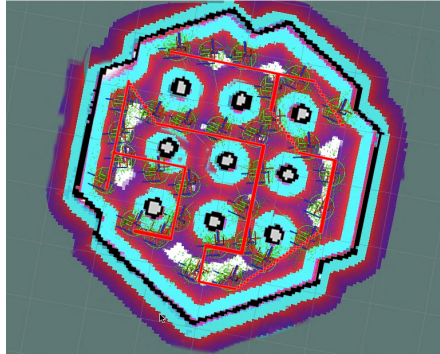


Figure 6.6: Turtlebot3 simulation. The initial map merged with all the waypoints reached. This color code is the same as the figure before, adding the path created by the planner in red.

When analyzing the data provided by RVIZ it can be seen that there were two paths from one waypoint to another that took more time than the rest. The two paths are shown in fig.6.7. Looking at the figures it can be seen that it happened in two corners, which makes sense since these corners have an obstacle nearby making them a narrow passage for the robot. With four identical corners in this environment, it would be logical to think that it happened in all four corners, but the reality is that it only happened in two. This is because of the path planning the planner did, in the path, fig.5.12, the robot only needs to go through the path on the corner in two of the corners. One thing that can be optimized in future work is to have the planner take all these variables into account when planning

the path. In this experiment, looking at the map with the waypoints and at the path made by the robot it can be concluded that the robot got to all the waypoints set by the planner. The robot covered an area of 31 waypoints distributed in approximately $19,76m^2$ in 5 minutes, which is an average of $0,25min/m^2$ or $15secs/m^2$. This rate is with almost no stops at the waypoints because the planner only defines a pose and doesn't choose an orientation, in each waypoint the robot always orients itself in the same direction wasting some time at each waypoint with the turn. In real-world use, the robot will have to stop at each waypoint. Also, the 31 waypoints set by the planner were enough to disinfect the whole area, but, the path proposed by the order of the waypoints given by the planner, is not the most optimized and could be in a more sequential order, which is another variable to take into account in future work.

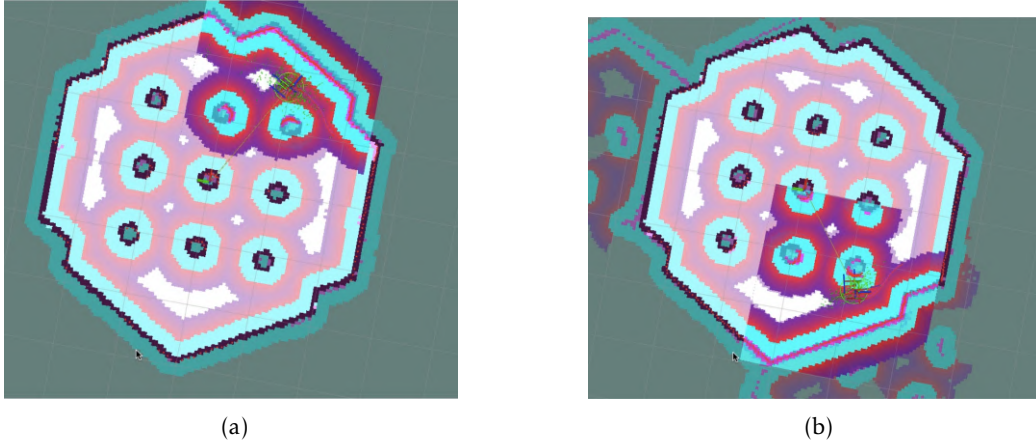


Figure 6.7: Turtlebot3 simulation. Two paths that took longer than others to conclude. This figure has the same color code explained before.

6.2 Gazebo World

In this experiment, the first step was to create a world in Gazebo, in this case, the world was a small room with some diagonal walls and some obstacles in the middle to represent the hospital reality, where there will be obstacles in the rooms and not all rooms will be rectangular. The obstacles in the room represent a hospital bed and some instruments in the room. This world is approximately $17,44m^2$. The world created can be seen in the figure below, fig.6.8, where the walls are represented in a lighter grey, the obstacles in a slightly darker grey, the robot is the small cylinder in the middle of the room and the blue area represents the area covered by the LiDAR sensor when the robot is in that position.

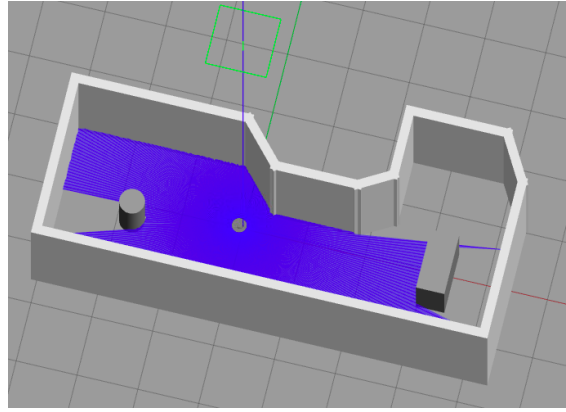


Figure 6.8: Gazebo world. Walls are represented in a lighter grey, the obstacles in a slightly darker grey, the robot is the small cylinder in the middle of the room and the blue area represents the area covered by the LiDAR sensor when the robot is in that position.

The gazebo world created requires a map, as previously mentioned in this dissertation. To do this, SLAM Toolbox can be used to map the world, as explained before. The gazebo simulation was launched with the new world in it and the robot started to map the world. Fig.6.9 shows the robot mapping the world. After scanning and mapping the whole room, the final result of the map was saved as a .yalm and as a .pgm file, and it can be seen in fig.6.10.

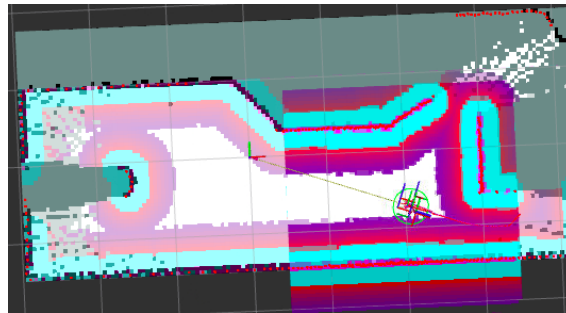


Figure 6.9: Gazebo simulation. Representation of the robot mapping with SLAM Toolbox. This figure has the same color code explained before.

As discussed in the previous experiment, the parameters for the planner to plan the waypoints depend on the size of the robot and the strength of the UV-C lights. In this experiment, the parameters of velocity were the same as in the previous experiment, the planner parameters chosen were 0.5m distance between waypoints and 0.2m distance to the walls. With these parameters, the planner choose 40 waypoints as shown in figure 6.13.

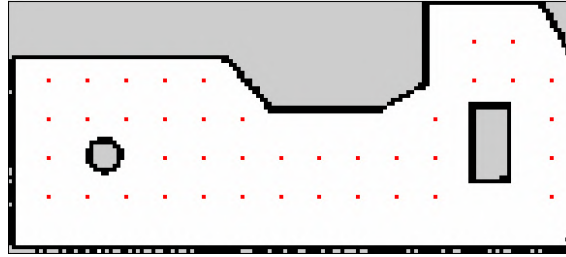


Figure 6.13: Gazebo simulation. Map with waypoints, parameters 0.5m distance between waypoints and 0.2m distance to the walls. The color code is the same as the previous map created by SLAM Toolbox, with the waypoints marked in red.

In this case, assuming the same robot, meaning the same distance to the wall, if the lights were stronger, allowing a distance between waypoints of 0.7m, the planner chose 23 waypoints, as shown in figure 6.14. If the lights were weaker and the distance between waypoints could only be 0.2m, then the planner chose 256 waypoints, as shown in figure 6.15.

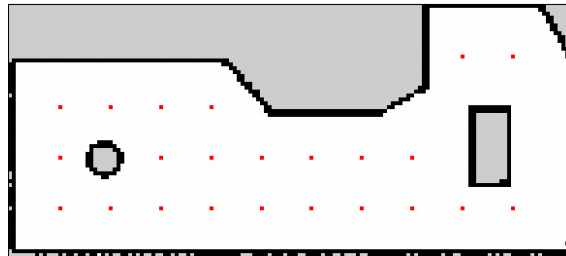


Figure 6.14: Gazebo simulation. Map with waypoints, parameters 0.7m distance between waypoints and 0.2m distance to the walls. The color code is the same as the previous map created by SLAM Toolbox, with the waypoints marked in red.

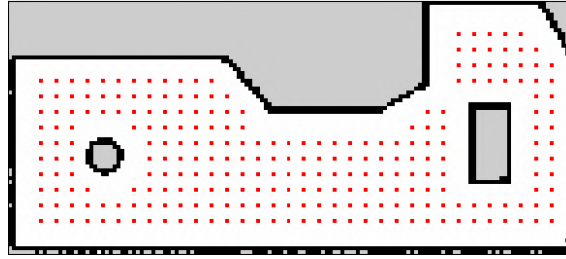


Figure 6.15: Gazebo simulation. Map with waypoints, parameters 0.2m distance between waypoints and 0.2m distance to the walls. The color code is the same as the previous map created by SLAM Toolbox, with the waypoints marked in red.

On the other hand, if the lights allowed a distance of 0.5m between waypoints but the robot was bigger, then the distance to the wall would have to be 0.5m and the robot wouldn't be able to reach some parts of the room, as seen in figure 6.16.

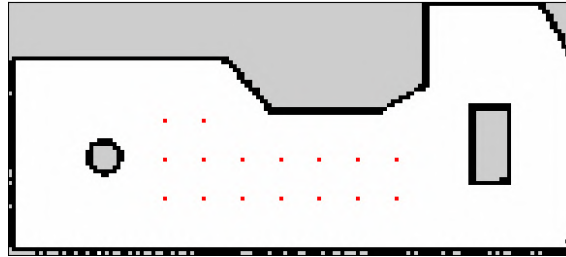


Figure 6.16: Gazebo simulation. Map with waypoints, parameters 0.5m distance between waypoints and 0.5m distance to the walls. The color code is the same as the previous map created by SLAM Toolbox, with the waypoints marked in red.

When the robot reached the last waypoint on the waypoints list, set by the planner, a message of success was printed. In fig.6.17 it can be seen all the waypoints that the robot reached. Merging that figure with the one printed in the beginning with the map and the path for the robot to follow and the result is fig.6.18.

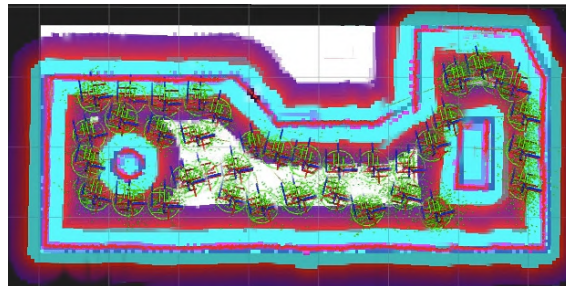


Figure 6.17: Gazebo simulation. Map with all the waypoints reached. This figure has the same color code explained before.

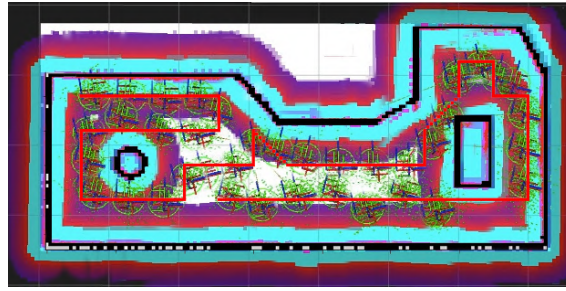


Figure 6.18: Gazebo simulation. The initial map merged with all the waypoints reached. This figure has the same color code explained before.

In this experiment, and with the help of the figures, it can be concluded that the robot reached all the 40 waypoints set by the planner and that, as in the previous experiment, the waypoints were enough to disinfect the whole room. The robot covered the whole area of the map, $17,44m^2$, in 5 minutes and 15 seconds, resulting in an average of approximately $0.3min/m^2$. As explained in the previous experiment, the robot is not yet programmed to stop at each waypoint. Nevertheless, as also explained before, because the planner only sends a pose to Nav2, rather than an orientation, in some waypoints the robot takes longer because it assumes one orientation and it has to rotate.

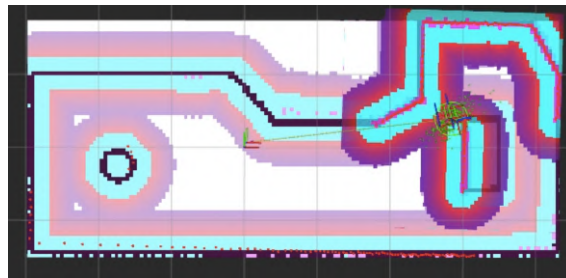


Figure 6.19: Gazebo simulation. Difficult passage. This figure has the same color code explained before.

After analyzing the data from RVIZ, one thing that was also very important in this experiment and is represented in figure 6.19, is that from one waypoint to another waypoint the robot could not execute the path planned, because the first path sent by Nav2 was too narrow. Still, because it is being used Nav2 it recalculated a new path and could go through the narrow passage from one waypoint to another. This process took 8 recoveries before Nav2 planned a path that the robot could use, and this path required the robot to backtrack slightly in order to pass the narrow passage. In total, the path between these two waypoints took 23 seconds, which is more than double the average of the rest of the other paths. In this experiment, the order of the waypoints is significantly better than in the previous experiment, which is a good thing because this room is a lot more like a hospital room than the one in the turtlebot3 experiment. But, it can also be improved to avoid a narrow path like the one described before.

CONCLUSIONS AND FUTURE WORK

This chapter presents a summary of this dissertation's achievements and some future work that can be done to improve and optimize this dissertation.

7.1 Conclusions

This dissertation presented a new model for disinfecting hospitals. A system was designed in order to make an autonomous robot able to disinfect a hospital floor. The model used ROS2 for the robot operating system, Nav2 for the navigation of the robot, SLAM Toolbox to map the hospital floor, a mobile robot, a LiDAR sensor, and UV-C lights. This system is able to map a hospital floor every time it is needed, even though the hospital rooms should always have the least amount of objects, for the robot to be able to move all around the room and reach every part of the room. Furthermore, because rooms don't always have the same objects, the robot will need to map the room from time to time. The robot, with the help of the planner created, is able to set waypoints in the map and order them in a way that makes a logical path. With these waypoints set and ordered, the robot uses Nav2 to plan the path between waypoints and control the robot in a way that it goes to every waypoint set by the planner.

With the experimental results shown before, it is concluded that the system is able to set the waypoints in the map in a way that disinfects the whole room. It is also concluded that the path calculated by the planner when ordering the waypoints has some failures yet, in most cases, makes an efficient path to get to all the waypoints. Not only that but the robot can always receive the waypoints and reach each and every waypoint in the established order. As discussed before, some paths between waypoints are harder than others and sometimes the robot takes longer to go from one waypoint to another, but with the help of Nav2, the robot would always find a path that allowed it to reach the next waypoint.

In the experiments shown before it can be seen that in one the robot covered in average $1m^2$ in 0.25min, a room with $19,76m^2$, and in the other in 0.30min, in a room with $17,44m^2$. This shows that even though the times are very similar, with the same speed

parameters, different rooms will have different speeds. This depends on the number of waypoints set by the planner, and the difficulty of the path between waypoints - as mentioned before, some paths between waypoints took longer than others.

Regarding what was proposed in the introduction, the last experiment should have been a real-world experiment with a real robot, but to the date of this dissertation, the robot wasn't ready to be used in the experimental results.

7.2 Future Work

Based on the proposed model, in this system, some points are missing and others can be improved. One of the things that is missing is making the robot stop for a certain time at each waypoint and turning the lights on and off. Although this is simple to add, there was not enough time to implement it in this thesis. As previously mentioned, the ordering planner could be optimized as it has some failures in some environments and doesn't always order the waypoints in the most efficient way for the robot to navigate. In the future, this planner should not only order the waypoints in a more direct way, but should also take into account that some paths take longer than others and that sometimes it's easier to go around an object than try to go through a narrow passage. Other than this, there are some things that can be added to the system that will make it easier and more efficient to program. One of them is implementing this system in a behavior tree so if the hospital needs to have the robot doing different tasks, such as transporting material, it will be a lot faster to program it and use the different tasks. For the map block, there could be lasers in each room that would scan the room after each patient left, and have some algorithm update the room layout in the map. This would provide information to the robot without forcing it to re-map the room each time, thereby saving time, robot battery, and also not having the disturbance of a robot moving around in a hospital floor without the need to disinfect.

BIBLIOGRAPHY

- [1] J. M. Lourenço. *The NOVAthesis L^AT_EX Template User's Manual*. NOVA University Lisbon. 2021. URL: <https://github.com/joaomlourenco/novathesis/raw/master/template.pdf>.
- [2] J. M. Boyce. “Modern technologies for improving cleaning and disinfection of environmental surfaces in hospitals”. In: *Antimicrobial Resistance & Infection Control* 2016 5:1 5 (1 Apr. 2016), pp. 1–10. ISSN: 2047-2994. DOI: [10.1186/S13756-016-0111-X](https://doi.org/10.1186/S13756-016-0111-X). URL: <https://aricjournal.biomedcentral.com/articles/10.1186/s13756-016-0111-x>.
- [3] H. Choi et al. “Classical and alternative disinfection strategies to control the COVID-19 virus in healthcare facilities: a review”. In: *Environmental Chemistry Letters* 19 (3 June 2021), p. 1945. ISSN: 16103661. DOI: [10.1007/S10311-021-01180-4](https://doi.org/10.1007/S10311-021-01180-4). URL: [/pmc/articles/PMC7820091](https://pmc/articles/PMC7820091) / [https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7820091/](https://pmc/articles/PMC7820091/?report=abstracthttps://www.ncbi.nlm.nih.gov/pmc/articles/PMC7820091/).
- [4] W. A. Rutala, M. F. Gergen, and D. J. Weber. “Room Decontamination with UV Radiation”. In: *Infection Control & Hospital Epidemiology* 31 (10 Oct. 2010), pp. 1025–1029. ISSN: 0899-823X. DOI: [10.1086/656244](https://doi.org/10.1086/656244). URL: <https://www.cambridge.org/core/journals/infection-control-and-hospital-epidemiology/article/abs/room-decontamination-with-uv-radiation/3B9FF95E5148FAC143D5342B89E0A791>.
- [5] *Culligan Commercial & Industrial Ultraviolet Water Treatment | Reynolds Culligan*. URL: <https://www.reynoldsculligan.com/culligan-commercial-industrial-ultraviolet-water-treatment/> (visited on 09/21/2023).
- [6] *TRACS Air Purifiers UV-C Replacement Lamps*. URL: <https://tracspurifiers.com/products/air-purifier-uv-c-replacement-lamps> (visited on 09/21/2023).
- [7] *How Do UV Phone Sanitizers Work? What You Need to Know*. URL: <https://www.businessinsider.com/guides/tech/how-do-uv-phone-sanitizers-work> (visited on 09/21/2023).

- [8] *SteriPro*. URL: <https://www.uvc-solutions.com/> (visited on 09/22/2023).
- [9] *Spot*. URL: <https://shop.bostondynamics.com/spot> (visited on 02/26/2022).
- [10] *A9-E / AUV / Autonomous Underwater Vehicle | Eca Group*. URL: <https://www.ecagroup.com/en/solutions/a9-e-auv-autonomous-underwater-vehicle> (visited on 02/26/2022).
- [11] E. W. Dijkstra et al. "A note on two problems in connexion with graphs". In: *Numerische mathematik* 1.1 (1959), pp. 269–271.
- [12] P. E. Hart, N. J. Nilsson, and B. Raphael. "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.
- [13] *Global Planner*. URL: http://wiki.ros.org/global_planner (visited on 02/26/2022).
- [14] S. Quinlan and O. Khatib. "Elastic bands: Connecting path planning and control". In: *[1993] Proceedings IEEE International Conference on Robotics and Automation*. IEEE. 1993, pp. 802–807.
- [15] D. Fox, W. Burgard, and S. Thrun. "The dynamic window approach to collision avoidance". In: *IEEE Robotics & Automation Magazine* 4.1 (1997), pp. 23–33.
- [16] C. Rösmann, F. Hoffmann, and T. Bertram. "Integrated online trajectory planning and optimization in distinctive topologies". In: *Robotics and Autonomous Systems* 88 (2017), pp. 142–153.
- [17] L. Calderita et al. "A new human-aware robot navigation framework based on time-dependent social interaction spaces: An application to assistive robots in caregiving centers". In: *Robotics and Autonomous Systems* 145 (2021), p. 103873.
- [18] K. Bungert et al. "Human-Aware Robot Navigation Based on Learned Cost Values from User Studies". In: *2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN)*. IEEE. 2021, pp. 337–342.
- [19] L. Bruckschen et al. "Human-aware robot navigation by long-term movement prediction". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 11032–11037.
- [20] S. Yao et al. "Crowd-Aware Robot Navigation for Pedestrians with Multiple Collision Avoidance Strategies via Map-based Deep Reinforcement Learning". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 8144–8150.
- [21] K. Li, Y. Lu, and M. Q.-H. Meng. "Human-Aware Robot Navigation via Reinforcement Learning with Hindsight Experience Replay and Curriculum Learning". In: *arXiv preprint arXiv:2110.04564* (2021).
- [22] P. T. Singamaneni, A. Favier, and R. Alami. "Human-Aware Navigation Planner for Diverse Human-Robot Interaction Contexts". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 5817–5824.

- [23] R. Alami et al. “Hateb-2: Reactive planning and decision making in human-robot co-navigation”. In: *2020 29th IEEE international conference on robot and human interactive communication (RO-MAN)*. IEEE. 2020, pp. 179–186.
- [24] P. C. Gøtzsche. “Niels Finsen’s treatment for lupus vulgaris”. In: *Journal of the Royal Society of Medicine* 104 (1 Jan. 2011), p. 41. ISSN: 01410768. DOI: [10.1258/JRSM.2010.10K066](https://doi.org/10.1258/JRSM.2010.10K066). URL: <https://pmc/articles/PMC3014565/><https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3014565/>.
- [25] M. Raeiszadeh and B. Adeli. “A Critical Review on Ultraviolet Disinfection Systems against COVID-19 Outbreak: Applicability, Validation, and Safety Considerations”. In: *ACS Photonics* 7 (11 Nov. 2020), pp. 2941–2951. ISSN: 23304022. DOI: [10.1021/ACSPHOTONICS.0C01245](https://doi.org/10.1021/ACSPHOTONICS.0C01245). URL: <https://pubs.acs.org/doi/full/10.1021/acsphotonics.0c01245>.
- [26] ROS: Home. URL: <https://www.ros.org/> (visited on 04/28/2023).
- [27] Why ROS 2? URL: www.openrobotics.org (visited on 04/28/2023).
- [28] Understanding nodes — ROS 2 Documentation: Humble documentation. URL: <https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html> (visited on 04/28/2023).
- [29] Understanding topics — ROS 2 Documentation: Humble documentation. URL: <https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html> (visited on 04/28/2023).
- [30] Understanding services — ROS 2 Documentation: Humble documentation. URL: <https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Services/Understanding-ROS2-Services.html> (visited on 04/28/2023).
- [31] Understanding parameters — ROS 2 Documentation: Humble documentation. URL: <https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Parameters/Understanding-ROS2-Parameters.html> (visited on 04/28/2023).
- [32] Understanding actions — ROS 2 Documentation: Humble documentation. URL: <https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Actions/Understanding-ROS2-Actions.html> (visited on 04/28/2023).
- [33] T. Bailey and H. Durrant-Whyte. “Simultaneous localization and mapping (SLAM): Part II”. In: *IEEE Robotics and Automation Magazine* 13 (3 Sept. 2006), pp. 108–117. ISSN: 10709932. DOI: [10.1109/MRA.2006.1678144](https://doi.org/10.1109/MRA.2006.1678144).
- [34] NAVIGATION | English meaning - Cambridge Dictionary. URL: <https://dictionary.cambridge.org/dictionary/english/navigation> (visited on 07/11/2023).
- [35] TRACER – Agilex Robotics. URL: <https://global.agilex.ai/products/tracer> (visited on 06/01/2023).

- [36] *Autonomous Mobile Robots* | Teradyne. URL: <https://www.teradyne.com/robotics/autonomous-mobile-robots/> (visited on 09/28/2023).
- [37] *Advanced Factories 2019: Autonomous Mobile Robots for industry*. URL: <https://blog.pal-robotics.com/advanced-factories-autonomous-mobile-robots/> (visited on 09/28/2023).
- [38] *UTM-30LX* | *Products List* | *Scanning Rangefinder* | *Distance Data Output* | *UTM-30LX* | HOKUYO AUTOMATIC CO., LTD. URL: <https://www.hokuyo-aut.jp/search/single.php?serial=169> (visited on 07/11/2023).
- [39] “TUV Amalgam XPT systems TUV 180W XPT SE UNP/20”. In: (2023). URL: www.lighting.philips.com.
- [40] *Nav2 — Navigation 2 1.0.0 documentation*. URL: <https://navigation.ros.org/> (visited on 07/12/2023).
- [41] S. Macenski and I. Jambrecic. “SLAM Toolbox: SLAM for the dynamic world”. In: *Journal of Open Source Software* 6 (61 May 2021), p. 2783. DOI: [10.21105/JOSS.02783](https://doi.org/10.21105/JOSS.02783). URL: https://github.com/SteveMacenski/slam_toolbox.

