POLITECNICO DI TORINO

Master's Degree in Mechatronics Engineering

Master's Degree Thesis

# Path planning algorithm for an autonomous air sanitizing mobile robot in indoor scenarios

Supervisors

Prof. Marcello CHIABERGE

Dott. Chiara BORETTI

Dott. Andrea EIRALE

Dott. Mauro MARTINI

Candidate

Carlo BARBARA

April 2023

# Summary

Robotics is now a well-established but constantly evolving sector which has reached multiple field ranging from large manipulators for industry applications to smarter mobile robots for housework that requires greater human interaction capabilities. An interesting branch in the robotics world is the Service Robotics, whose main purpose is to help people and improve their quality of life. Mobile robots are widely used in this area, they have to carry out tasks that require them to navigate in known and unknown spaces, thus are of paramount importance the abilities to measure the space around them with different kind of sensors (Lidar, camera, TOF, ...) and to try to have a perception of their position in the 3D-space (Odometry). In the past two years, one of the biggest problems that has involved our society is Covid-19, a dangerous virus that makes the air its main vector of contagion.

The objective of this thesis work, which is part of a collaboration between the PIC4SeR (Politecnico di Torino Interdepartmental Center for Service Robotics) and the Innovation Center of Intesa Sanpaolo, is to develop an autonomous mobile platform, equipped with CO2 sensor and an air sanitizer, capable of performing the sanitization of the air in a working environment (offices, meeting rooms, etc.) to limit the risks of Covid-19 contagious in indoor scenarios.
In details, a simulated environment with different rooms and corridors was created. Then, after creating a detailed map of the working environment, a path planning algorithm has been developed using ROS2 and NAV2. This custom algorithm optimizes the path that the robot has to follow when it moves from one room to another, but the algorithm selects also the shortest path to perform the sanitization process of a room. In addition, in the path planning algorithm, the state of charge of the robot's battery has been taken into account as well as the possible presence of closed doors between different rooms. The final goal is to perform a shorter and better optimized sanitization.

# Acknowledgements

ACKNOWLEDGMENTS

# Table of Contents

# List of Tables

# List of Figures

# Acronyms

**QoS**

Quality of Service

**SOC**

state of charge

**ToF**

Time of Flight

**UV**

ultraviolet

**AI**

Artificial Intelligence

**PAS**

photoacoustic spectroscopy

**FoV**

field of view

# Chapter 1

# Introduction

## 1.1 Overview

In recent past years the innovation of robotics brings to develop even smarter and more preforming robots. The aim of robots is to help, or even replace, people in their hard, risk or/and repetitive jobs, preventing them from overexerting themselves and achieving more efficiency and repeatability in work processes.
Also the interaction robots-humans is slightly increased, especially for mobile robots that manage tasks in public environments and with a large presence of people. With this new kind of smarter robot become very impotant terms like safety, because no one should be hurt by these machines, and security, due to the fact that many robots work in private houses and factories, hospitals or other critical places, for this reason is fundamental people who do not have the required permissions should not have access to the control and data of these machines.

There are two macro categories of robots: **industrial robots** and **service robots**. The first type of machines are most used for manufacturing, to handle tasks that require big efforts like moving heavy loads or apply big forces and torques. They are usually fixed or only able to move on a track.
The second type of robots are used in a vast number of different fields.complex scenarios. The complexity of these robot is due to the huge variety of working scenarios, and to the close contact with humans, situation that occurs in many of these robot's tasks [1].

$(FONTEhttps : //upload.wikimedia.org/wikipedia/commons/e/ee/Pepper_{t}he_{R}obot.jpg)$
This master thesis project was born from a collaboration between the Politecnico di Torino Interdepartmental Center for Service Robotics (PIC4SeR) and the Innovation Center of Intesa Sanpaolo.
The aim of the project is to realize an autonomous mobile service robot that is able

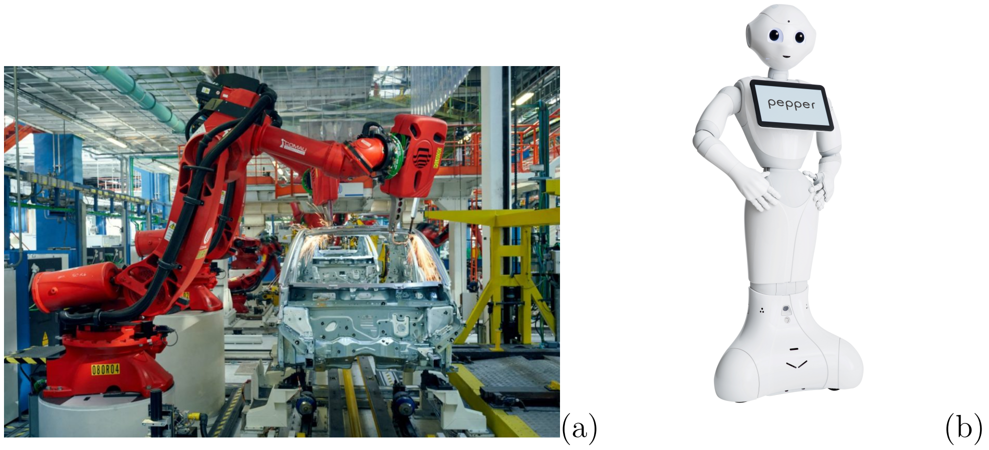

**Figure 1.1:** (a) COMAU industrial robot in a FCA production line [2] (b) Pepper humanoid robot from SoftBank Robotics

to perform the air sanitation of an indoor environment in total autonomy, even in presence of people.

## 1.2 Service Robotics

Service robotics is wide utilized in many fields: agriculture, medicine and health, space, underwater exploration, and so on. Service robots very often have to operate in the presence of people, therefore they must be able to autonomous navigate in an indoor environment, to detect obstacles in their path trough the usage of multiple sensors and/or cameras, having an idea of its position in the 3D space and be aware of the possible presence of people in the environment, being capable of working around them and adjust its task and behaviors based on this, without interfering or being a danger for people (FONTE https://www.market-prospects.com/articles/what-is-a-service-robot).

(FONTE IMMAGINE a https://www.cyberweld.co.uk/robots-in-agriculture-and-farming) (FONTE IMMAGINE b https://www.hydro-international.com/content/news/under robot-helps-complete-scientic-survey-of-maldives-waters) (FONTE IMMAGINE c https://viterbischool.usc.edu/news/2021/03/how-robots-stay-on-track-in-giant-automated-warehouses/) (FONTE IMMAGINE d https://www.bostondynamics.com/atlas)

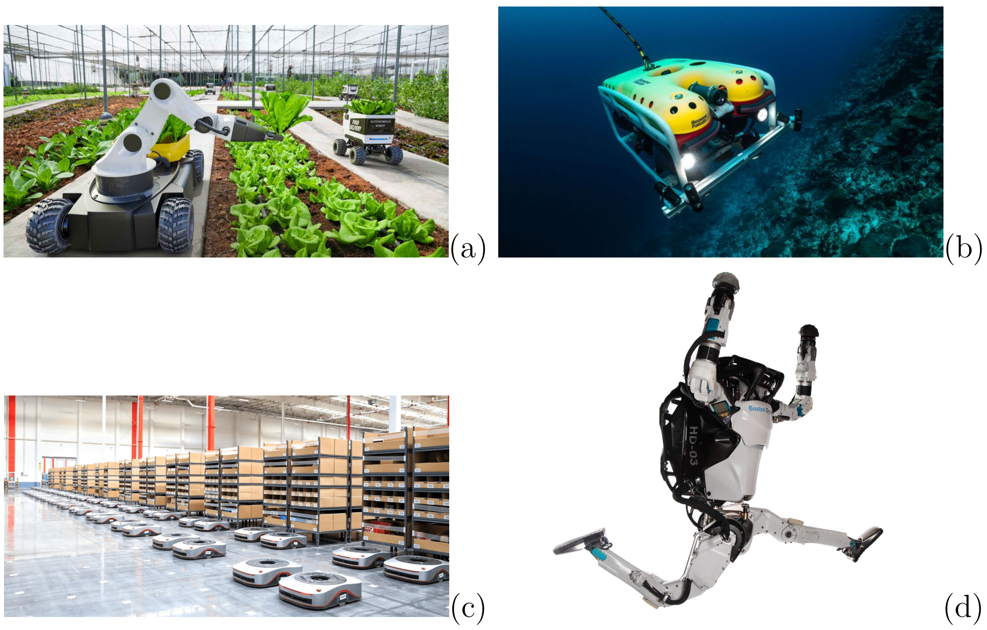

**Figure 1.2:** (a) Service robots used in agriculture (b) Robot for underwater explorations (c) Robots working in automated warehouse (d) Atlas anthropomorphic robot from Boston Dynamics

## 1.3 Indoor navigation

In the majority of cases, robots have to navigate in an environment to complete their tasks. Navigation is a real complex feature for a robot, because it is made itself by many other sub-problems like **maps and environment representation**, **planning techniques** and **odometry and perception**. Solving these problems is equivalent to answer questions like: where am I? Where are other objects respect my position? How can I reach that certain position? (FONTE http://didawiki.cli.di.unipi.it/lib/exe/fetch.php/pro/pro10-b5-navigazionerobot.pdf)

### 1.3.1 Maps

Maps are the representation of the real world in a logical domain. There are two main types of maps:

- **geometric maps** are based on an absolute reference frame and respect environment's dimensions and coordinates in this frame. The most used are the *occupancy grid* (MORE OF THIS IN NAV2 CHAPTER) and the *geometric description* (FONTE IMMAGINE https://www.semanticscholar.org/paper/A-random-finite-set-approach-for-dynamic-occupancy-Nuss/10bdf1a15b6c95f031038314ef63c2c

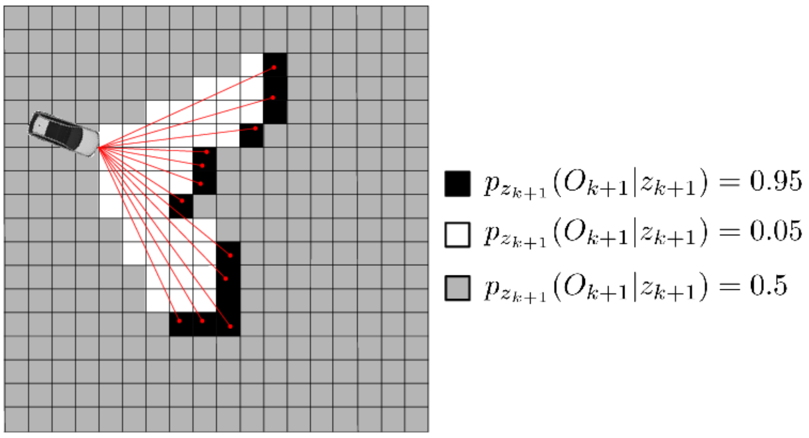
■ $p_{z_{k+1}}(O_{k+1}|z_{k+1}) = 0.95$
□ $p_{z_{k+1}}(O_{k+1}|z_{k+1}) = 0.05$
▨ $p_{z_{k+1}}(O_{k+1}|z_{k+1}) = 0.5$

**Figure 1.3:** Occupancy grid example

- **topological maps** represent the key points of the environment on the basis of the relations between them and their functionalities. An example is the *node graph*. (FONTE IMMAGINE https://cs.slu.edu/ esposito/teaching/1080/webscience/graphs.html)

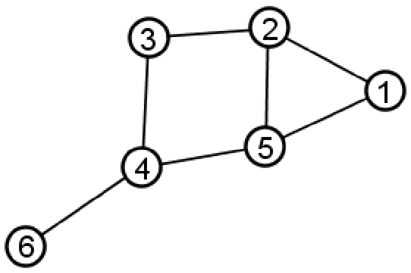

**Figure 1.4:** Nodes graph example

## 1.3.2 Path planning

Path planning is fundamental to compute the path the robot must follow to reach a final goal position. There are many path planning algorithms, both for geometrical and topological maps.
For the first type of maps, a very common strategy is the one called **roadmap**, that consists in the connection of some points of the map creating a network. Based on the roadmap, the *visibility graph* is a technique that uses a graph that connects the robot starting position, the goal position and all the obstacles vertices. The final path is computed retrieving the shortest path that connects the starting position

to the final one without crossing the obstacles' polygons.
Other methods not based on the roadmap, but pretty similar, are the *cell decomposition* and the *potential fields* method, that is the one used in the ROS costmap. (MORE OF THIS LATER IN CHAPTER...) For topological maps, the path planning algorithms are similar to the previous, but in this case the map itself is the graph from which the path is computed. The algorithm that is implemented also in this thesis work is the Dijkstra algorithm (MORE OF THIS IN CHAPTER...). (FONTE

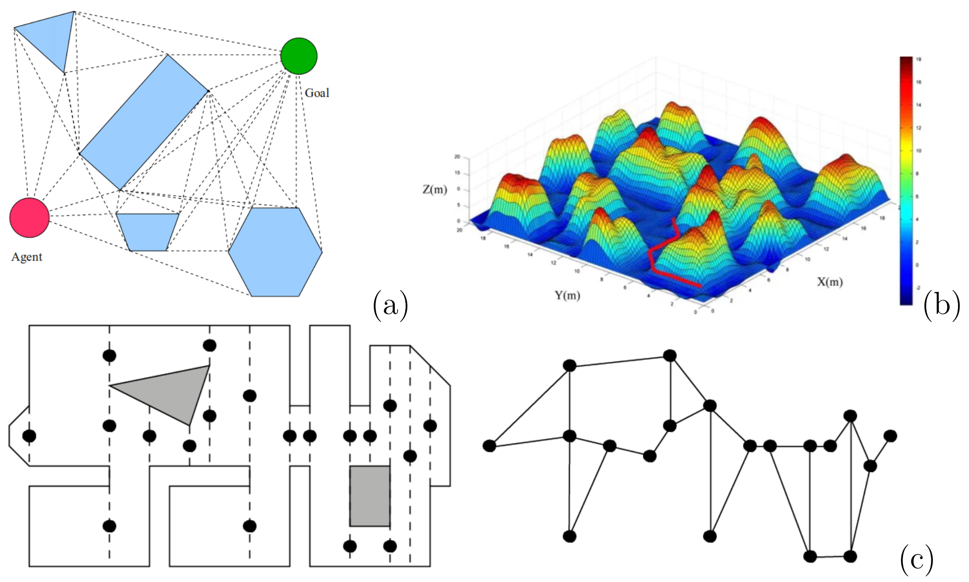

**Figure 1.5:** (a) Visibility graph technique example (b) Potential fields technique example (c) Cell decomposition technique example

IMMAGINE a $https : //www.researchgate.net/figure/Visibility-Graph-the-dashed-lines-represent-candidate-paths-for-the-vehicle-and-the_{f}ig2_325371124$) (FONTE IMMAGINE b https://www.semanticscholar.org/paper/Exact-cell-decomposition-of-arrangements-used-for-Sleumer-Tschichold-G(FONTE IMMAGINE c https://medium.com/@rymshasiddiqui/path-planning-using-potential-field-algorithm-a30ad12bdb08)

### 1.3.3 Path following

After the path planning algorithm computes points where the robot should pass, then the path following technique is responsible for the **trajectory planning** and the **motion control**: the first phase computes the trajectories in the reference frame that connects two points of the planned path, than the motion control, through automatic control and inverse cinematic techniques, computes the voltages required from the motors to follow the desired velocities and accelerations to be able to correctly follow the reference trajectories.

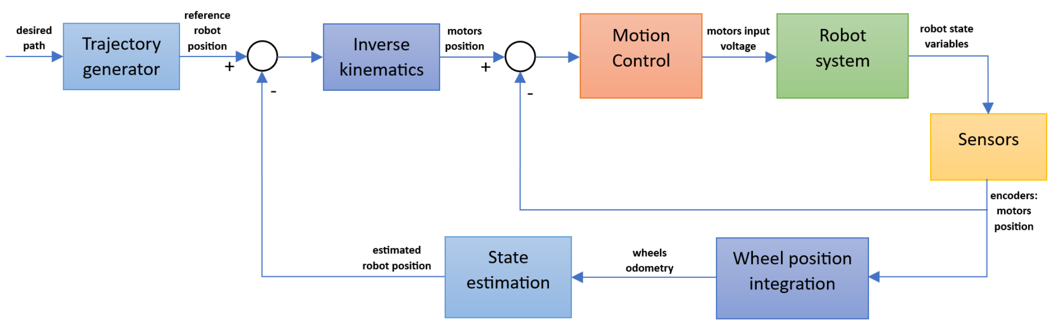

**Figure 1.6:** Path following control scheme example with wheels odometry

The path following depends from robots' characteristics, for example wheels type and configurations that determine the movements that the robot can perform, i.e. the trajectories that it is able to follow.

### 1.3.4 Perception and Odometry

Perception is performed by sensors, they collect different kind of data from the environment in order to detect the different objects the robot can meet during the navigation.
Sensors like the **encoders** are *proprioceptive* sensors, they measure robot's proprieties, in this case wheel angular position and velocity.
**LiDAR** or **camera** instead, are *exteroceptive* sensors, retrieving information from the space around the robot.
With both these type of sensors is possible to have an idea of the robot position in the environment: integrating the measured speed movements (**odometry**, a possible source of error is the wheel slip), or detecting recognizable objects with a camera and analyzing how their positions change in time (**visual odometry**, camera image distortion can bring errors). $(FONTEhttps://cgi.csc.liv.ac.uk/~trp/COMP329_files/COMP329-2017-Lecture6b.pdf)$ $(FONTEhttps://en.wikipedia.org/wiki/Visual_odometry)$

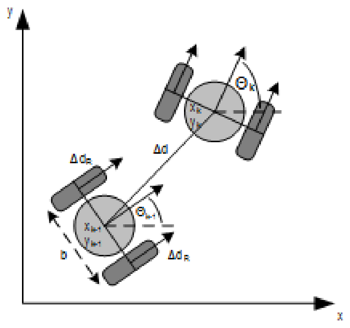

**Figure 1.7:** Example of wheels odometry

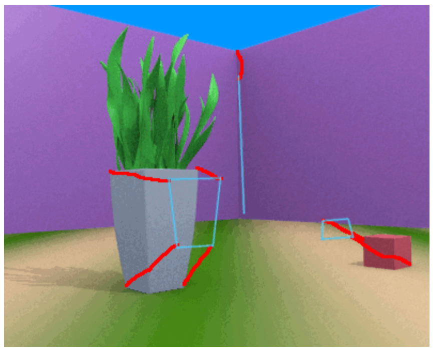

**Figure 1.8:** Example of visual odometry

(FONTE IMMAGINE ODOMETRY https://hackaday.io/project/158496-imcoders/log/147068 robot-odometry) (FONTE IMMAGINE VISUAL ODOMETRY è LA STESSA DEL PARAGRAFO)

## 1.4 Air sanitizing robots and related works

Robots for sanitation and disinfection already exist and are widely used in some hospital environments to ensure a safe environment for patients without exposing people to any risk.
In Slovenia, two hospitals uses robots, equipped whit UV lights, whose purpose is to sterilize the hospital. (FONTE https://www.intelligentcitieschallenge.eu/covid-19-good-practices/disinfection-robots-hospitals)

(FONTE IMMAGINE https://www.healthcareitnews.com/news/emea/dubai-health-authority-deploys-robots-disinfect-facilities) After the 2020 COVID-19 pandemic crisis, the development of this type of robot had a sharp peak, to make not only hospital and healthcare environments safe, but also normal offices, schools and other workplaces.

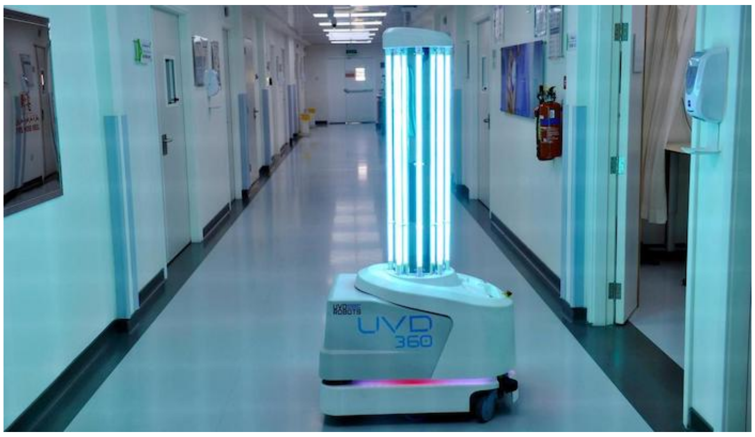

**Figure 1.9:** UVD 360 from UVD Robots

There is also a project for an autonomous indoor cleaning robot, that perform the sanitation of the air with a HEPA air filtration system and in addition a UV-C light, taking the air from the side of the robot and expelling it from the top to obtain a better distribution of the sanitized air flux. Then, knowing the air flow rate of the robot and the dimension of the environment, an AI algorithm assign a working area to each of the operating robot tacking into account the time elapsed since the area's last sanitation [**sanitationrobot**].
With this thesis project instead, the intention is to have one or more robot that perform a smart sanitation, measuring the $CO_2$ quantity in the air in one or more points of a room, and perform the sanitation if the measure is higher than a certain threshold level. Following this methodology the air sanitation, that is an expensive operation in terms of battery power, will be performed only when needed.

Why $CO_2$? Because COVID-19 is a very contagious virus and its transport vector are very small liquid particles that people can spread in the air coughing, sneezing, speaking, or simply speaking ($FONTEhttps://www.who.int/emergencies/diseases/novel-|$ $coronavirus-2019/question-and-answers-hub/q-a-detail/coronavirus-$ $disease-covid-19-how-is-it-transmitted?gclid=Cj0KCQjww4-$ $hBhCtARIsAC9gR3Yrnra0AlUhaDA7K4WkmKrkj7CVpca7URaM9WUKlSCo49KIlqCG$
During these actions, people also emit $CO_2$ into the air, so there is a probabilistic match between the relative quantity of $CO_2$ and fluid particles that may contain COVID-19 virus.

# Chapter 2

# ROS framework and navigation stack

## 2.1 ROS and ROS2

ROS (Robot Operating System) is an open source software for the management of robotics tools, components and interfaces, developed by Willow Garage in 2007. Whit ROS is simple to put in communication the different parts that make up a robot, like actuators, sensors and control units, and this greatly facilitates the development of applications for robots.
(FONTE IMMAGINE e non https://www.ros.org/) ($FONTEhttps : //it.wikipedia.org/wiki/R$

**Figure 2.1:** ROS logo [3]

Despite being innovative and useful, however, ROS did not satisfy many market demands, among which the most important were real-time operations, safety and security. To meet these requests, ROS2 was developed, his alpha version was created in 2015 and the first official release was published in 2018. ROS2 was created to enhance the features that made ROS so widely used, and to fill the gaps with customers demands, for these reasons there are some main differences between the two versions.
The main differences are the **master node** that in ROS is required before running any other node because it is like a server for them but in ROS2 is not needed

because every node has the ability to discover the others in the network. A great improvement in terms of compatibility and simplification in programming is the insertion of a new library sub-layer, the *rcl* base library, on which the client libraries like *rclcpp*, *rclpy*, *rcljava* and others, are then built, that instead in ROS those libraries are completely independent and features developed for one library is not compatible with the others.
Another great innovation is the implementation of **lifecycle nodes**: this new type of nodes are a 4-states machine able to decouple every node phase (setup, run...) from the others. Further useful items are **launch** files that let to launch more nodes at the same time, setting parameters and other options, and the introduction of the **QoS** to make nodes' communications more reliable.
All these new features contribute to make ROS2 a simpler and more inclusive tool. (FONTE https://roboticsbackend.com/ros1-vs-ros2-practical-overview/)

The most important elements of the ROS2 network are:

- **nodes**: principal part of ROS nodes are the all different processes used for every part of the robot. For example there is a ROS node for processing the measures of a sensor, or another one that is used to control motors actuators, a node for algorithm implementation that performs computation. Every ROS node is able to send and receive messages to and from other nodes. In this way is possible for example for a sensor node to sends measurements to a control unit node that then sends commands to an actuator nodes.

- **messages**: nodes exchange messages between them to communicate. There are standard messages like *Int16*, *Float64*, and so on, or it is possible to create custom messages with more fields of different datatype.

- **topic**: ROS topics use a publisher/subscriber communication protocol, so any node can publish on a topic and the other nodes can receive that message subscribing to that topic. It is a many to many communication. A topic can receive/send only messages of the same type, this type is defined when the topic is declared in nodes' code. ($FONTEhttps : //docs.ros.org/en/foxy/_images/Topic-|$ $MultiplePublisherandMultipleSubscriber.gif$)

- **services**: services uses a request/reply communication protocol. Topics allow a continuous communication between nodes, services instead provide data only when a client sends a request. Service clients can be more than one, but there is only one service server for a service. ($FONTEhttps :$ $//docs.ros.org/en/foxy/_images/Service - MultipleServiceClient.gif$)

- **actions**: ROS2 actions are a type of communication designed for long running tasks. It is a mix of service and topic communications, the first one is used for

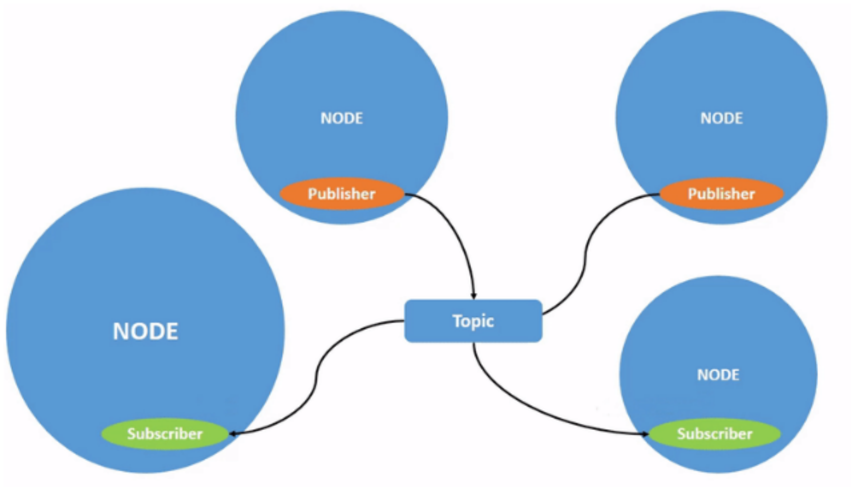

**Figure 2.2:** nodes and topic example

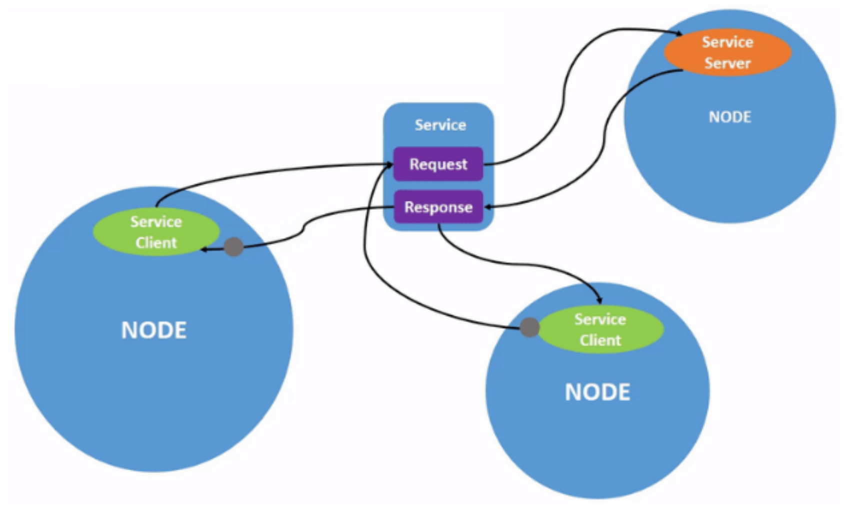

**Figure 2.3:** service example

the *goal service* where the client node send the goal and the server one acknowledges it, then through the *feedback topic* the server node updates the client on the status of the task, in the end the server sends the final result using the *result service.* $(FONTEhttps : //docs.ros.org/en/foxy/_images/Action - SingleActionClient.gif)$

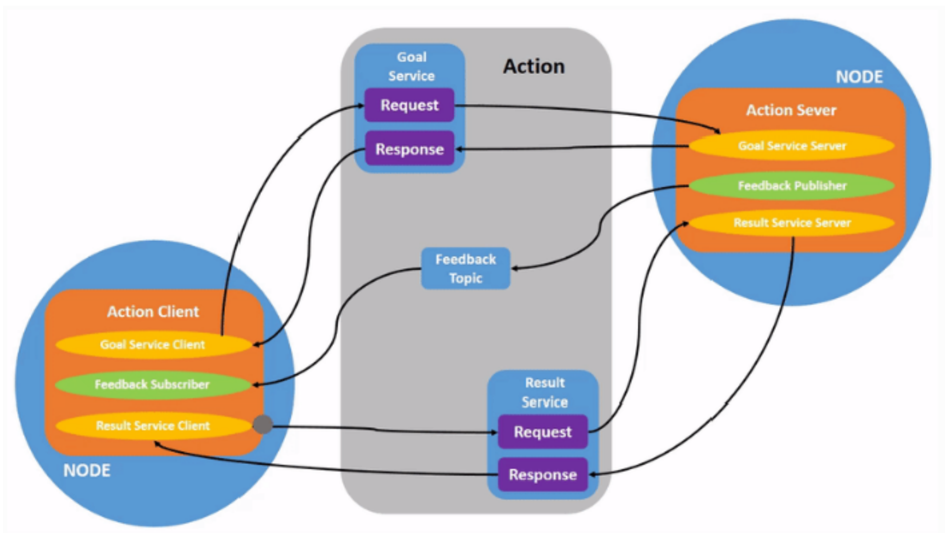

**Figure 2.4:** action example

## 2.2 Navigation Stack and NAV2

ROS Navigation stack is a group of ROS packages made to simplify robot navigation tasks. For navigation is intended the management of all the nodes responsible for the odometry and motion control, so the robot can reach a position goal from its starting position. Of course this is a tool available only for mobile robots.
The evolved version of Navigation Sack, based on ROS2, is NAV2. $(FONTEhttps:$

**Figure 2.5:** NAV2 logo

$//navigation.ros.org/_static/nav2_logo.png)$
NAV2 uses the new communication innovation of ROS2, the action services. As said before, action services are designed for long time tasks, and navigation is one

of these, because it is an operation that depending from velocity limitations and map dimensions, could keep the robot busy for a time interval long enough to need feedback on the operation and eventually also to cancel the goal.
For robot navigation is widely used the **Behaviour Tree** concept that consists in tree structures of tasks which can be reused for different contests and this allows to manage a vast spectrum of robot cases that would otherwise be very difficult and complex using a state machine that would present an excessive number of states and transitions.
NAV2 uses different action server nodes to manage all navigation tasks like the **planner**, responsible for the computation of the path based on the received goal, the **controller**, that basically has the task of following up the trajectory computed by the planner. (FONTE https://navigation.ros.org/concepts/index.htmlconcepts)

The concept of **costmap** is very important in the navigation context. It is a representation of the environment in a 2D grid of cells. Each cells contains a value that represents the occupancy probability, using this cells the planner is able to compute the path with the minimum "*cost*".

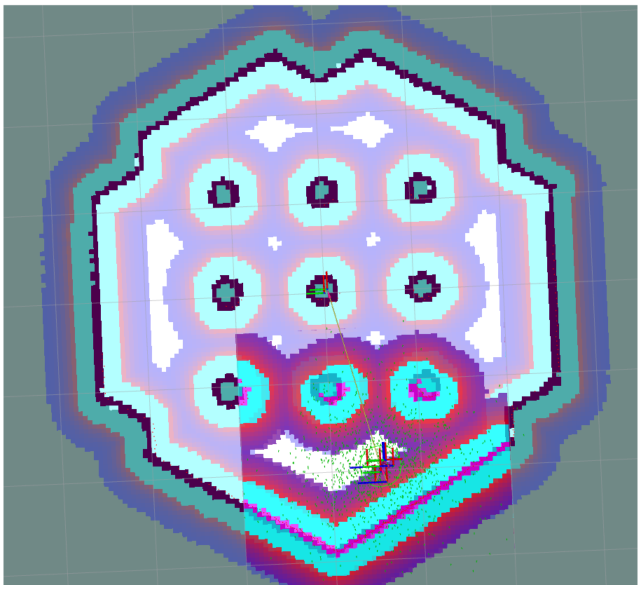
**Figure 2.6:** costmap of the NAV2 turtlebot simulation

The **global costmap** is derived directly from the map file and it covers all the map extension, the **local costmap** instead has a shorter coverage and it is derived from robot's sensors like LiDAR and camera.

## 2.3 Gazebo

Gazebo is a 3D simulation environment optimized for robotics simulation with apposite libraries and tools for sensors simulation and actuators control. It allows to easily creates robots mechanical components, building models and other objects, and simulate their kinematics and dynamics with a discreet level of precision $(FONTEhttps : //en.wikipedia.org/wiki/Gazebo_simulator)$. (FONTE IMMAG-

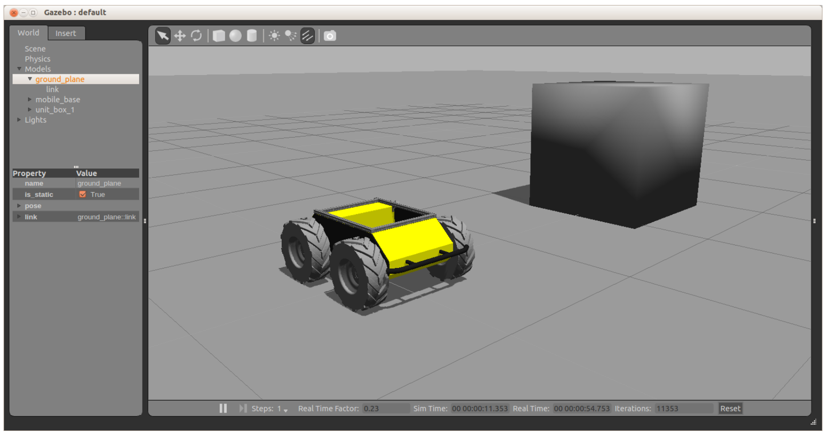

**Figure 2.7:** Husky robot in a Gazebo simulation

INE https://www.clearpathrobotics.com/assets/guides/noetic/ros/Drive

## 2.4 RViz

RViz (ROS Visualization) is a graphic interface that allows to show on screen all the information contained into different topics, and its especially used to represent the content of navigation topic, like map information, all the different layers of global and local costmaps, goal pose, computed path to follow, all the reference frames (absolute and relative), LiDAR and other sensor measurements and so on. (FONTE IMMAGINE http://www.xaxxon.com/documentation/view/oculus-prime-ros-tutorial-navigation-using-rviz)

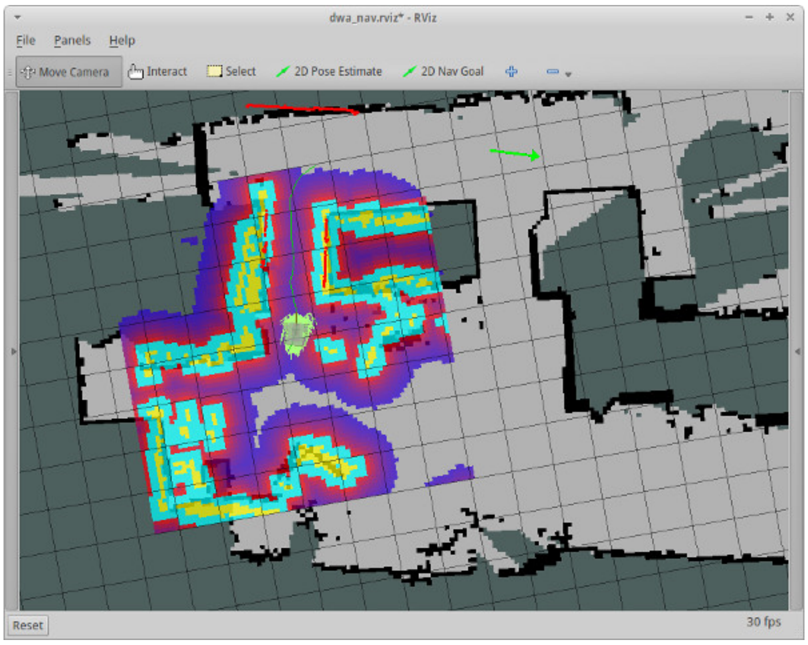

**Figure 2.8:** RViz graphic interface representing map and costmap

Chapter 3

# Robot Hardware

The path planning algorithm of this thesis project was developed on the bases of a robot prototype developed in PIC4SeR, that consists of a set of various components and sensors that let the robot performs all the different functions needed during its tasks.
The sanitation of an indoor environment requires mobility and odometry capabilities for navigation, a device able to filter ans sanitize the air, camera for perception and object detection, and a sensor able to detect also transparent obstacles. Everything it will be manage from a PC mounted directly on the robot and equipped with Ubuntu OS and ROS2.

## 3.1 KUKA youBot mobile base

KUKA youBot mobile base is responsible for the movement of the robot and contains also the embedded PC. It is a differential mobile robot provided of four motor-encoders connected to four omniwheels.
Thanks to these particular wheels, the robot can perform also a translation on the lateral direction enhancing the robot's mobility and this is a very important feature for a mobile robot operating in a human environment.

(KUKA youBot Datasheet: $http://www.youbot-store.com/wiki/index.php/YouBot_Detailed$
(IMMAGINE 3D $https://www.researchgate.net/figure/Youbot-from-Kuka-|$
$It-provides-unconstrained-planar-movement-with-use-of-mecanum-$
$wheels_fig2_316366419$)

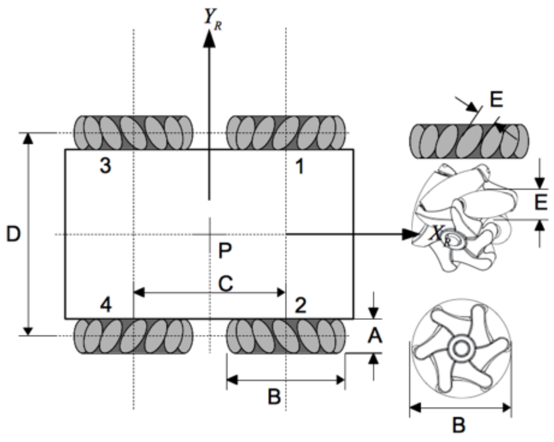

**Figure 3.1:** KUKA youBot base plant and omniwheels

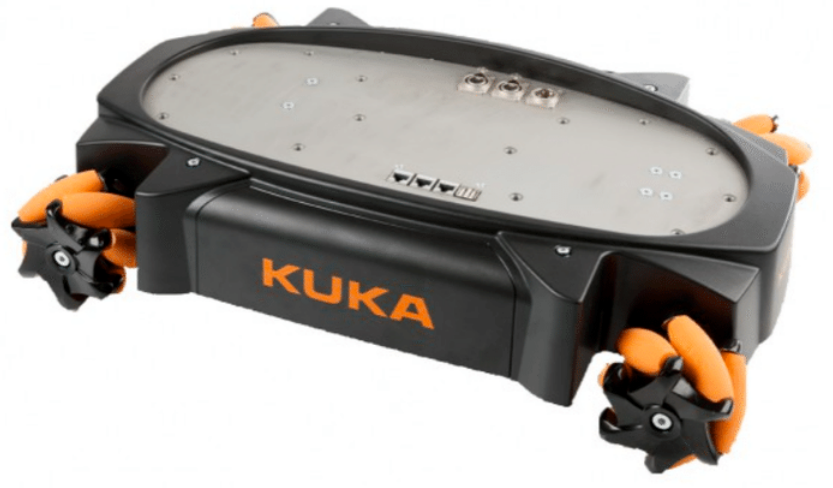

**Figure 3.2:** KUKA youBot mobile base

## 3.2 VORTICE VORT ARIASALUS 200

The component used for the air sanitation and purification task is the VORT ARIASALUS 200, by VORTICE. It is mounted on the top of the KUKA youBot base and it is the largest, heaviest and most energy consuming component of the

prototype.
It has a 5-layer filter unit:

- plastic resin dust filer
- ISO COARSE 90% (G4)
- EPA E11 filter
- anti-TVOC filter
- photocatalysis device

The firsts layers retains all the air impurities while the last component is the one able to eliminate bacteria and virus, in particular SARS-CoV-2. This technology, combained with a periodic air exchange, guarantees a high level of cleanliness and air healthiness. (FONTE https://www.vortice.it/it/trattamento-aria/depuratori/centralizzati/25044)
It has three different velocity level for the

**Figure 3.3:** VORT ARIASALUS 200

## 3.3 XENSIV PAS CO2 sensor

Due to the high consuming sanitation device, it is not possible to continuously performing the air sanitation procedure while the robot is moving (it would add

also the motor-encoders consumption), otherwise robot's autonomy would be too short. It is more efficient to sanitize only when needed, according to measurements of the level of present in the air.
For this reason a specific sensor is required. XENSIV™ PAS CO2 sensor from Infineon Technologies is a very compact sensor whit a high measurement accuracy. One of his potential applications is air quality monitoring ventilation control, that is exactly the scope of this sensor for this project. (FONTE e Datasheet https://www.infineon.com/cms/en/product/sensor/co2-sensors/pasco2v01/)

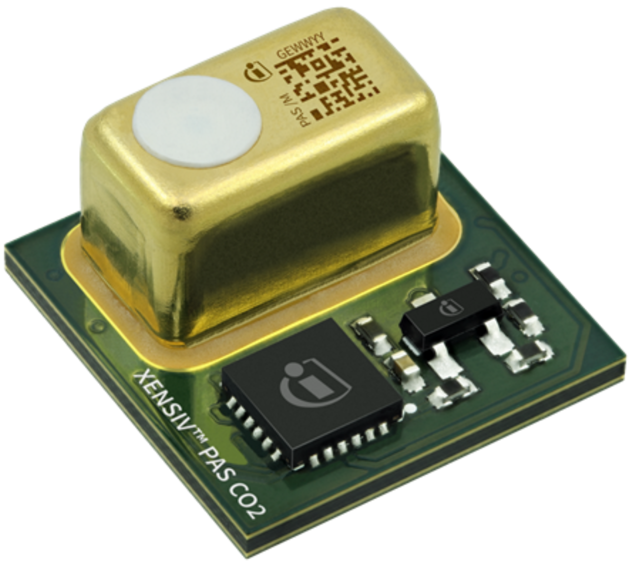

**Figure 3.4:** XENSIV PAS CO2 sensor

## 3.4 RPLIDAR A2 Laser Range Scanner

RPLIDAR A2 Laser Range Scanner, produced by Slamtec,is a 360 LiDAR sensor that uses infrared laser light to get a 3D with a maximum range of 16 m and performs 8000 samples per time with a 10 Hz frequency. These specifics allows a precise detection of obstacles and walls during the robot navigation, and simplifies also the creation of the environment's map reducing the required robot's movements thanks to the wide range of the laser scanning.

$(FONTEIMMAGINESCANEPAGINASENSOREhttps : //www.slamtec.com/en/Li$
$(FONTEIMMAGINESENSOREhttps : //my.cytron.io/c-sensor/c-laser-range-finder/p-rplidar-a2m8-the-thinest-lidar)$

**Figure 3.5:** RPLIDAR A2 Laser Range Scanner

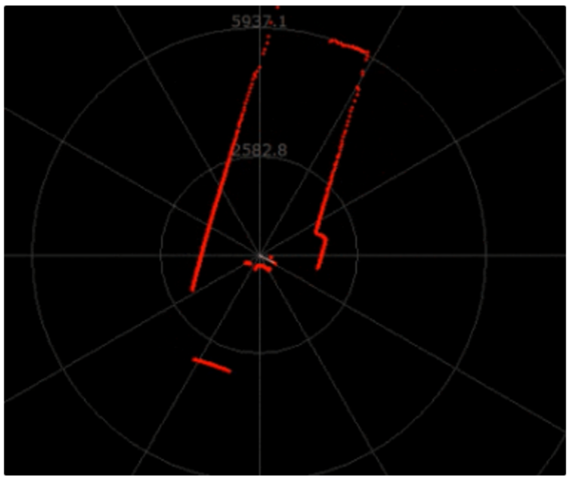

**Figure 3.6:** RPLIDAR environment scan

## 3.5 VL53L5CX Time-of-Flight sensor

LiDAR sensor is a very useful tool for obstacle detection thank to its wide range and precision. But it useless in the case obstacles are made of a translucent material because laser light beams pass through the obstacle.

In these cases a Time of Flight sensor is more useful. VL53L5CX sensor (produced by STMicroelectronics) performs an 8x8 multizone ranging (64 acquisitions per measurement) with a FoV of 65°diagonal square and a maximum detection range of 400 cm. ToF sensor is mounted on a P-NUCLEO-53L5A1 equipped with a ArduinoUnoR3 board. (FONTE VL53L5CX E IMMAGINE https://www.st.com/en/imaging-and-photonics-solutions/vl53l5cx) (FONTE P-NUCLEO-53L5A1, IMMAGINE DAL DATASHEET https://ie.rs-online.com/web/p/sensor-development-tools/2300082)

**Figure 3.7:** VL53L5CX Time-of-Flight sensor

## 3.6 Intel RealSense Tracking Camera T265

Is a small and low power consumption camera design for tracking, equipped with a V-SLAM algorithm (Simultaneous Localization and Mapping) that using camera vision and inertial measurements is able to track an unknown environment with high precision.
In this thesis project this camera is used for the visual odometry of the robot. (FONTE https://www.intelrealsense.com/tracking-camera-t265/)

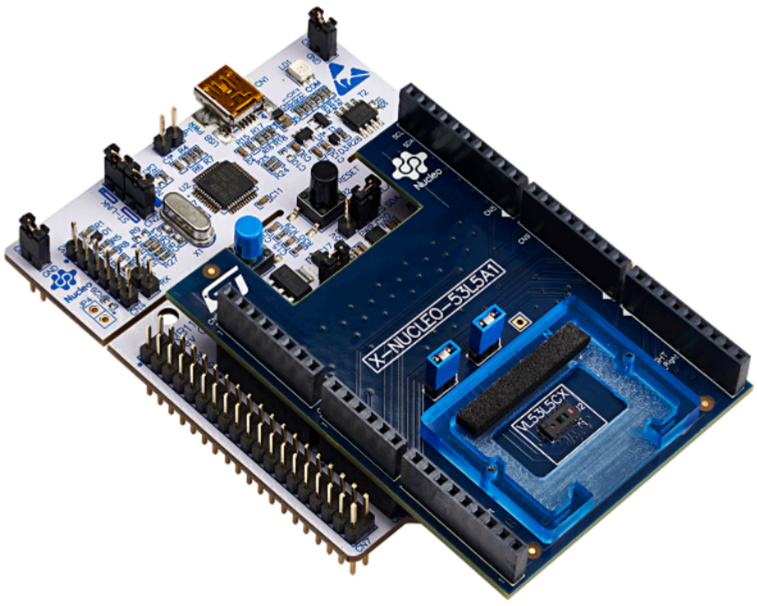

**Figure 3.8:** P-NUCLEO-53L5A1 Proximity Evaluation Board

**Figure 3.9:** Intel RealSense Tracking Camera T265

## 3.7 OAK-D camera

Luxonis OAK-D is made of three sensors: two grayscale cameras on both sides, and a color camera in the center. It has embedded AI algorithm to perform 2D and 3D object tracking using the grayscale cameras, and object detection with the color one in the center. This camera is used for the social navigation of the robot (not developed in this thesis work). (FONTE https://shop.luxonis.com/collections/usb/products/oak-d)

**Figure 3.10:** OAK-D camera

# Chapter 4

# Environment and sanitation procedure analysis

The first step for a path planning algorithm development is focusing on where robot will move around, in particular, the plant of the environment, how different rooms are linked between theme, the presence of doors and other obstacles in the path, and the material they are made of.

The materials play a very important role because in many cases robots use LiDAR sensor to detect obstacles around them, but if for example doors are made in glass, just like the case of this thesis work, LiDAR cannot see it (difficult also with a camera), so the robot does not know if there is an obstacle in front of it or not. On the contrary, a sensor that could detect transparent obstacles is the Time Of Flight sensor, but the range of its measures is more limited, and for this reason the navigation of the robot from the current position to the destination room must be split into more intermediate destinations that coincide with the exits/entrances of the room where a door could prevent the passage.

The aim of the robot is to sanitize the air in the different rooms of the environment. To optimize this task, it was tried to carry out the sanitation by making the robot move in different points of the room, measuring the quantity of $CO_2$ in the air and activating the fan of the sanitation device at different speeds based on those measurements. This technique could help to perform a more efficient sanitation of the environment, spending more resources where more needed.

## 4.1 Points and Exits

To manage this two main features of the robot's path planning two types of goal pose are considered:

- **Sanit Points** (in short *Points*): these points are the goal poses of the path planning algorithm. They are used to move the robot in one room in order to perform the sanitation process in more than one spot to ensure a more homogeneous air sanitation. In the algorithm these points are also responsible for the selection of the destination room that has to be sanitized;
- **Exits**: intermediate goal poses that are in the proximity of an exit from one room to another one. Exits are always present in pairs distributed in two adjacent rooms that can communicate between them.
  If there are two rooms, A and B, the notation for the Exits is: *Exit A.B* to indicate the exit from room A to room B, and *Exit B.A* for the opposite. Exit points number is equal to the double of the number of accesses between rooms.

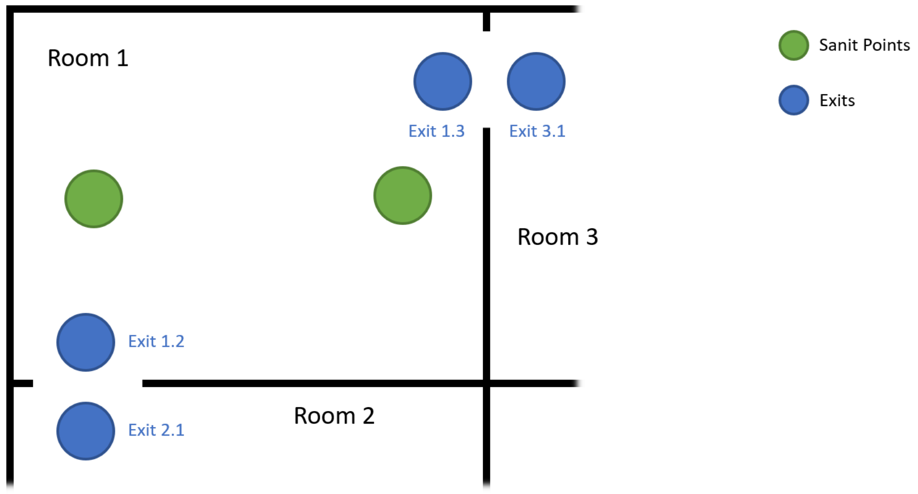

**Figure 4.1:** Points and Exits

## 4.2 Plant configurations

To explain better the logic and the ideas behind this path planning algorithm it is needed to introduce the concept of **Room**. In this thesis context, the meaning of Room is similar to the architectural one but does not coincide with it. It is a logical space in the working environment in which the robot has to complete a task.

Rooms aren't necessarily separated by walls, more Rooms can coexist in the same room of a plant, or one Room can be inside another one.

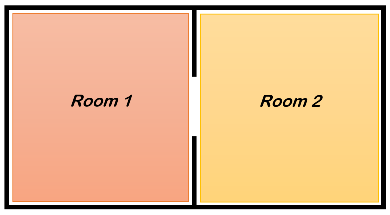

**Figure 4.2:** Two Rooms equivalent to architectonic rooms

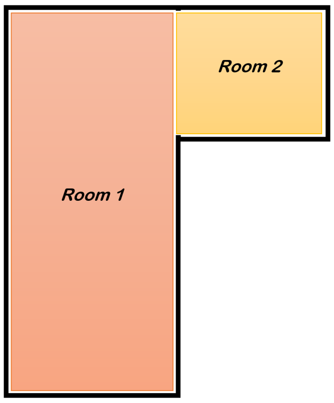

**Figure 4.3:** Two Rooms in a single architectonic room

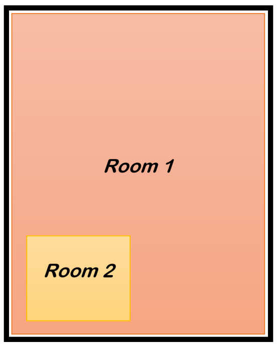

**Figure 4.4:** A Room inside another one

### 4.2.1 Single corridor and multiple rooms

The first idea of a map was a big corridor (Room 0) in which is located the spot for the battery recharging and has no Sanit Points inside.
Room 0 is connected to all the other rooms by only one access and these rooms don't communicates directly between them, so there are two Exits for each room (Room 0 to Room X and Room X to Room 0).
This kind of plant is the simplest, as every time a room needs to be sanitized the planning is a fixed pattern (assuming battery recharging is not needed):

1. *Navigate to the Exit 0.X*
2. *Check if the door is open*
3. *If open*:
   a. *Enter the Room X and perform sanitation*
   b. *Navigate to the Exit X.0*
   c. *Check if the door is open*
      - *Enter the Room 0*
4. *Go to the next exit*

(SOSTITUIRE L'ELENCO CON L'IMMAGINE DEL FLOWCHART)

In this case, if a door is closed, only two cases are possible:

- robot is blocked in one room and has to wait for the door to be open again;
- robot is in the corridor and skip to the next room;

Having to manage only this two cases is very simple from the logical point of view but it is not enough when the plant's shape become a little bit more complex.

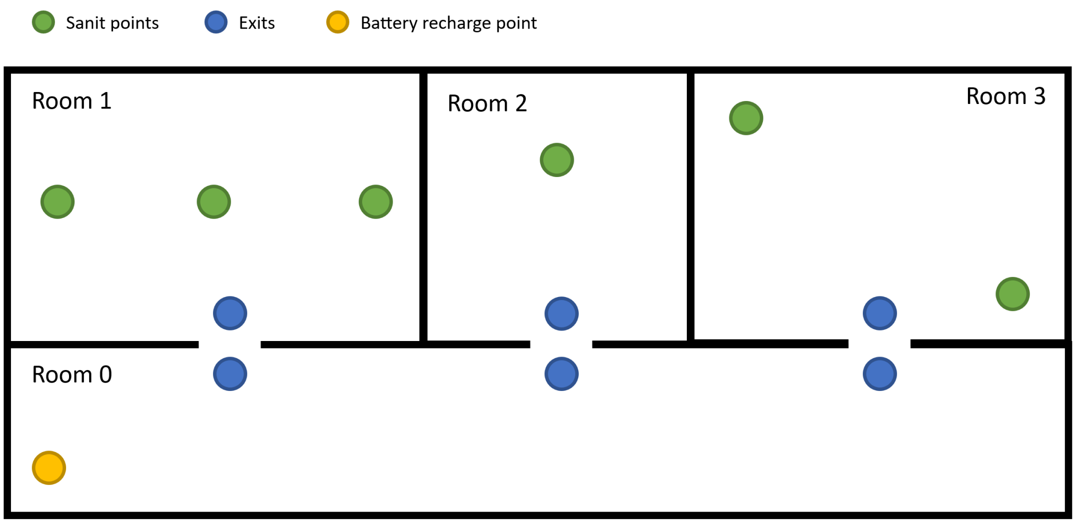

**Figure 4.5:** Corridor plant configuration

### 4.2.2 Communicating rooms

If in the plants there are some rooms that can communicate directly with others in addiction to Room 0, or that can communicate with other rooms but not with Room 0, the planning became more complex.

Is possible to notice that with this configuration, to reach the destination room, the robot may pass through other intermediate rooms without sanitize them, and the possible paths to reach the destination can be many.
In addiction if a door is closed doesn't prevent the robot to reach the destination because another path could be available, due to this it will become necessary to recompute the path including the information about the status of that closed door.

A possible solution to solve this problem is using a node graph that represents

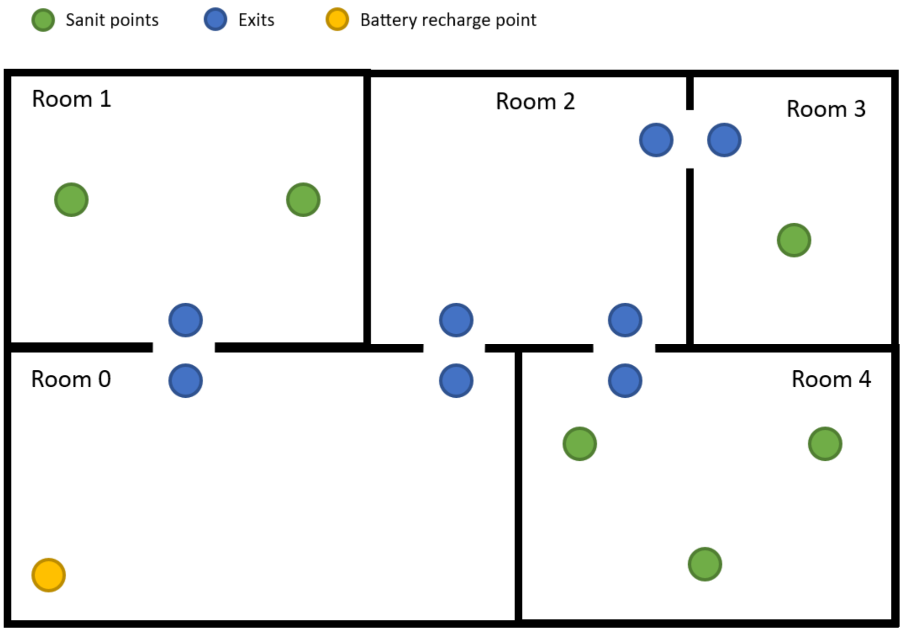

**Figure 4.6:** Complex plant with communicating rooms

the Points-Exits network and the Dijkstra algorithm to compute the shortest path from the start node to the destination node.

### 4.2.3 Multiple exits

In some plants could append that between two room there be more than one exit. In this case occurs to use a variable as an *exit index* that identifies uniquely the exit.
Therefore, if between Room A and Room B there are $n$ exits, there will be $2n$ exit poses, so Exits will become vectors: Exit A.B($i$) and Exit B.A($i$), where $i \in \mathbb{N} : i \in [0, n-1]$, is the exit index.

## 4.3 Navigation issues

In the most general plant type, with many rooms connected between them, eventually by more than one door, three main problems can arise which can interrupt the working schedule of the robot:

- **one navigation warning case**: this situation is verified when all the remaining not yet sanitized rooms are not reachable from the current robot position. It will try to reach these rooms for a maximum number of attempts, once

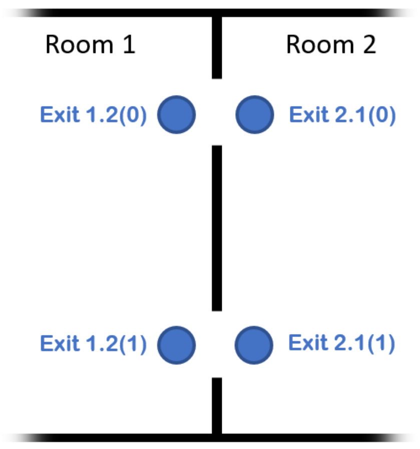

**Figure 4.7:** More than one exit between two rooms

this is exceeded, the robot will return to the charging station and then the sanitation schedule will be restarted. It is not a critical situation because the robot can still reach other rooms and the recharging position.

- **two navigation alarms cases**: these are the only two navigation cases in which the robot cannot proceed with his tasks:
  1. **robot is stuck in a room**: robot cannot reach any other destination because it is closed inside one room and cannot exit from any of the doors. This navigation alarm includes also the warning situation.
  2. **the battery recharge position is not reachable**: the robot needs to recharge its battery because the SOC is below a certain threshold but the pose where robot can recharge is not reachable.

In any other case, is possible to skip the current destination, trying to reach it again in another instant, an proceeding with the next one.

### 4.3.1 Robot stuck in a room

This situation is verified when a robot has ended a task in one room (battery recharging or air sanitation) and when the algorithm computes the next destination room, none of them is reachable.
More precisely this issue comes out when any point outside the room where the robot is located, is not reachable due to the combination of closed and open doors. Anyway, it does not represents a critical problem because robot can wait until a door of the room will be again open

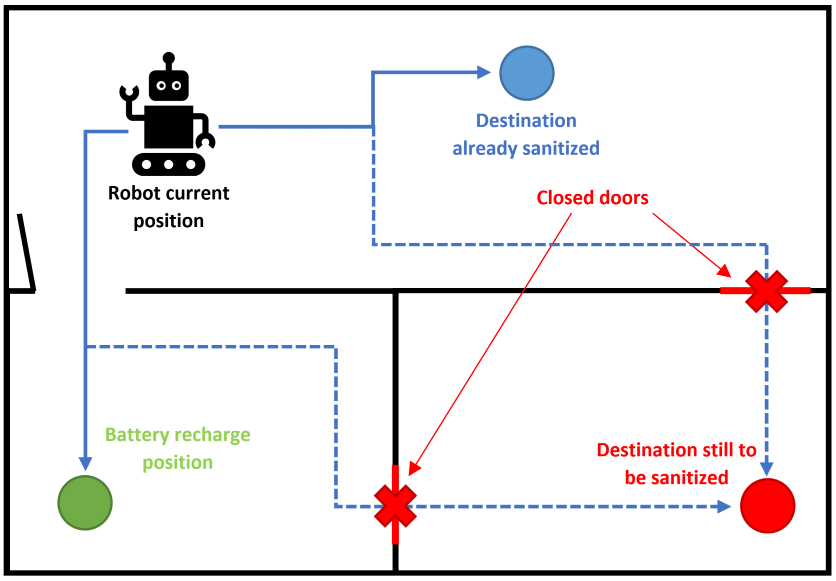

**Figure 4.8:** Example of navigation warning situation

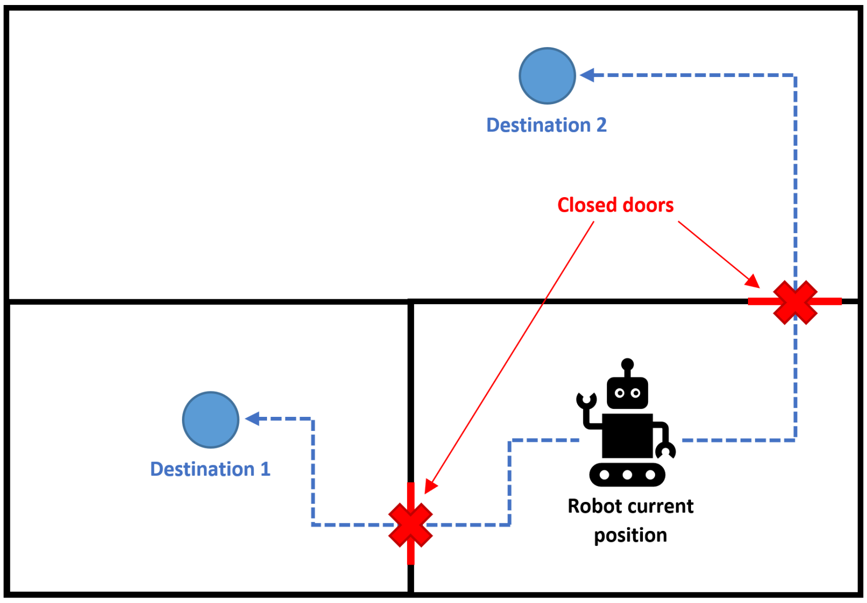

**Figure 4.9:** Robot stuck in one room

### 4.3.2 Battery recharge position unreachable

When the battery voltage goes under a certain threshold level, the path planning algorithm sets as next destination the recharging battery pose.
When this situation occurs but the destination is not reachable, it is the most

dangerous navigation issue because the robot could shout down and it will not be able to move anymore, being also an obstacle for people in the office.

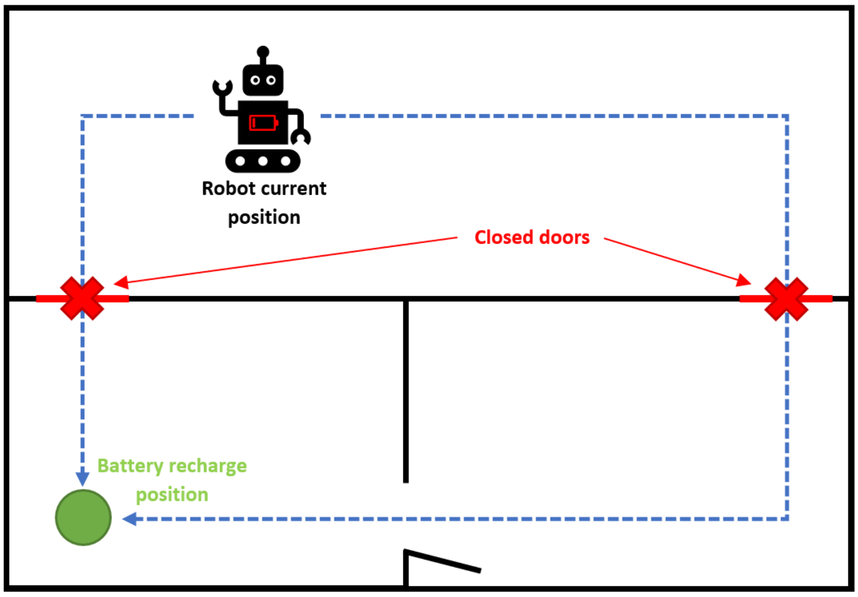

**Figure 4.10:** Low battery and battery recharge position not reachable

## 4.4 Sanitation planning

Regarding the sanitation task of the robot, it was said before that the aim of this process is to sanitize a room by filtering and purifying the air in one or more points, based on the size of the room and how much people use it. In the actual version of the algorithm, the Sanit Points of the rooms that have to be sanitized are predefined, but also other smarter methods for automatic Points selection could be implemented (MORE OF THIS IN FURTHER IMPROVEMENTS).

### 4.4.1 Sanitation schedule

When the robot enters in a room for the sanitation procedure, it has to move on each Sanit Point. Once it reaches one of them, it measures the $CO_2$ level in that point and, if needed, turn on the fan to a velocity level based on the measure and keeps it until a new $CO_2$ measure is below a certain level (SEE FURTHER IMPROVEMENTS). This task will go on until all the Points are sanitized (SEE CHAPTER 6.3) or when the robot continues to read high $CO_2$ quantities but reaches the maximum number of sanitation cycles of the room and it has to attempt

the procedure again in another moment.
It may happen that a rooms takes a lot of attempts to be sanitized, so there will be also a maximum number of attempts for which a room can be selected from the algorithm as the next destination.

# Chapter 5

# Network architecture

## 5.1 Layers

The path planning algorithm architecture is composed by two main layers:

- **Path planning algorithm**: developed in this thesis work, that manages all robot's behaviours and tasks and select all the destination and intermediate poses that the robot must reach using a **node graph**.
- **NAV2 Navigator Server**: used for the navigation inside the environment, taking in input the Schedule Node goal pose and monitoring the navigation and providing feedback, using a known map of the environment.

The path planning algorithm select the destination of the robot based on its current task and status, and then computes the path to reach it, creating a sequence of poses that the robot must follow.
It performs all this computations using a **node graph** built on the basis of the environment map.

The NAV2 Navigation Server is responsible for moving the robot inside a known map, using the behavior tree "*NavigateToPose*" that uses two action servers: "*ComputePathToPose*" and "*FollowPath*".
The first one compute the list of poses that let reach the destination trough the shortest path (similarly to what the path algorithm does), getting as input the starting and the destination pose of the robot. In this case there is a set of destination nodes that is provided by the path planning algorithm.
The second is responsible for the navigation from one pose to the next in the list previously computed, checking if the pose is correctly reached, and for the motion control of the robot.
Both this action server are dynamic because the path can be changed if some

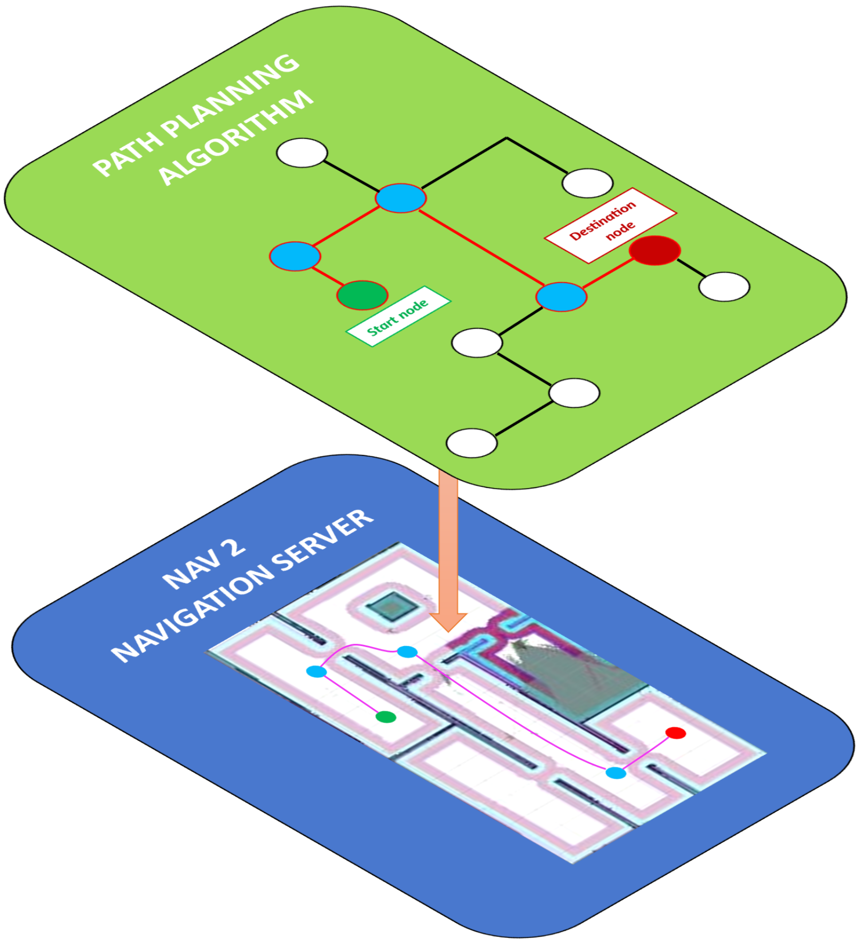

**Figure 5.1:** Layers structure

obstacle is detected during the navigation.

In a certain sense the path planning algorithm and NAV2 ComputePathToPose work in the same manner but the first operates on a higher level, basing his computations on the node graph, instead the second works on a lower level basing directly on the costmaps.

### 5.1.1 Map

The map is a fundamental components for the navigation. It consists of two files:

1. **yaml** file: contains different map information:
    - *image*: is the path to the pmg file.

  - *resolution*: indicates the resolution of the map [$meters/pixel$].
  - *origin*: used to locate the map respect to a fixed 2D reference frame, providing x, y and yaw of the lower-left pixel of the map respect to that frame.
  - *free thresh*: is the threshold level that determines if a pixel should be considered completely free or not. If the occupancy probability of a pixel is less than this threshold it will be considered as free.
  - *negate*: flag that inverts the gray scale evaluation of occupancy probability.

2. **pmg** file: this file contains an image in gray scale(white for completely free pixels, black for completely occupied pixels, and grey for unknown). Every pixel's gray scale level indicates the occupancy probability (or free probability if *negate* is equal to 1) of that pixel.

(INSERIRE IMMAGINE DEL FILE PMG) (FONTE: $https : //wiki.ros.org/map_{s}erverYAML_{f}$

Based on these files, NAV2 Navigation Server is able to derive the global costmap for navigation.

### 5.1.2 Node graph

In order to respect the navigations features expressed in the previous chapter (INSERIRE RIFERIMENTO AL CAPITOLO 4.1), a topological map, in particular a node graph, has been implemented, containing three different types of nodes: *Battery node, Points nodes, Exit nodes.*

#### Battery node

This one is the only node present in the Room 0, that is a fictitious Room inside the Room 1. The battery recharging node represent the starting pose of the robot, that is also the origin of map's reference system.
There are two conditions for the robot to return to this node once it has started:

- robot ended the sanitation schedule and has to wait to restart it again.
- the battery SOC is below the limit threshold and it is necessary to recharge.

Once on this node, the battery recharging process starts and the robot restarts again his schedule when the battery SOC is higher than a *full charge* threshold.
The links rule for this node is only one:

**1)** *the battery recharging node must be linked to all nodes of Room 1.*

(IMMAGINE BATTERY NODE E COLLEGAMENTI)

#### Point nodes

These nodes represent the Points of the map in the node graph. They are located in each room that must be sanitized and their number is variable from one room to another. Point nodes are used to select the final destination of the robot, in a first moment to choose which is the next room to sanitize, and then, during the sanitation process, to select the next Point in the room where measure the $CO_2$ level and, if needed, turn on the fan.
The links rule for this type of nodes is:

**1)** *each Point node must be connected to all the other nodes of the same room (also to the battery recharging node in the case of Room 1).*

This rule is needed for the room sanitation, in order to move the robot from one Point to the others, and let the robot moving to the exit of the next room of the path at the end of the sanitation process. (IMMAGINE POINT NODE E COLLEGAMENTI)

#### Exit nodes

Due to the need of perform a measure with a ToF sensor on the door position.
The links rules for this type of nodes are:

**1)** *each Exit node must be connected to all the other nodes of the same room (also to the Battery node in the case of Room 1).*

**2)** *each Exit node must be connected to the corresponding Exit node of the adjacent room.*

The first rule is to allow the robot to move towards a Point in order to start the sanitation process if the room is the one to be sanitize. Or, if not, the robot can go from an Exit to the next one continuing its path. The second one let the robot moving between two different rooms, and make possible simulating the door closure by cutting the link between the two Exits. (IMMAGINE EXIT NODE E COLLEGAMENTI)

## 5.2 Room class and data structure

In order to pass the goal pose to NAV2, the Schedule Node must be provided with a data structure containing all the useful information about the different rooms of the environment and the coordinates of the map's points of interest (Points and Exits).

A data structure is implemented in a Python class called "Room", to keep track of room state variables and collect pose coordinates and quaternion of Points and Exits. An object of this class is created for each Room on the map and than, a unique Room list *rooms* is created, which contains all the Room objects.

| **inputs list** | **data type** |
|---|---|
| *height* | $float16$ |
| *length* | $float16$ |
| *width* | $float16$ |
| *num people* | $uint8$ |
| *num points* | $uint8$ |
| *max sanit tries* | $uint8$ |
| *max attempts* | $uint8$ |
| *priority* | $float16$ |
| *num rooms* | $uint8$ |

**Table 5.1:** Room class inputs

### 5.2.1 Room Points and Exits

It has already been discussed what Points and Exits are and their importance path planning and robot navigation. These nodes, in addition to having a logical meaning for the algorithm, must be located in precise points within the map in order to be sent to the robot as destination goals. The generic data structure of a Room that contains Points and Exits coordinates and quaternion parameters are the Room class parameters *x, y, z, q0, q1, q2, q3* . Each of these parameters is implemented as a float16 list of lists, as can be seen from Table 5.3. In the first sub-list of each list are contained the pose coordinates of the Points of the room. After this one there are other $num\ rooms - 1$ lists, one for each Room of the map. These lists contain the pose parameters of the Exits between the RoomX and all the others. Obviously in every $i$-th struct, the *Exit i.i* field is left empty.

For algorithm implementation, Room0 was meant to be a fictitious Room inside Room1 (no Exit nodes for Room0) in which it is located only the Battery Recharge node. For this reason the Points fields of the Room0 coordinates data structure contain only one element, and all the Exit fields are empty (5.4).
With this data structure implementation, taking into consideration the struct of a generic Room$i$, to access a coordinates parameter, the $y$ for example, of the Exit $i.j(k)$ between the Room$i$ and another generic Room$j$, the command to use will be $rooms[i].y[j][k]$ ,

| **variable list** | **data type** | **description** |
|---|---|---|
| *height* | $float16$ | room height |
| *length* | $float16$ | room length |
| *width* | $float16$ | room width |
| *num people* | $uint8$ | number of people inside the room |
| *doar flag* | *bool* | 0: door is closed<br>1: door is open |
| *num points* | $uint8$ | number of Sanit Points oh the room |
| $x$ | $float16[\,]$ | x coordinate list of room's Points & Exits |
| $y$ | $float16[\,]$ | y coordinate list of room's Points & Exits |
| $z$ | $float16[\,]$ | z coordinate list of room's Points & Exits |
| $q_0$ | $float16[\,]$ | quaternion $q_0$ list of room's Points & Exits |
| $q_1$ | $float16[\,]$ | quaternion $q_1$ list of room's Points & Exits |
| $q_2$ | $float16[\,]$ | quaternion $q_2$ list of room's Points & Exits |
| $q_3$ | $float16[\,]$ | quaternion $q_3$ list of room's Points & Exits |
| *points sanit flag* | *bool*[ ] | flag list for sanitation tracking |
| *points order* | $uint8[\,]$ | list of the Points in order of sanitation |
| *room sanit flag* | $uint8$ | 0: not entered in the room<br>1: room non successfully sanitized<br>2: room successfully sanitized |
| *attempts* | $uint8$ | attempts performed to sanitize the room |
| *max attempts* | $uint8$ | maximum attempts to sanitize the room |
| *sanit tries counter* | $uint8$ | counter for room sanitation cycles tried |
| *max sanit tries* | $uint8$ | max room sanitation cycles for attempt |
| *priority* | $float16$ | default priority in destination selection |
| *reachable flag* | *bool* | 0: room not reachable<br>1: room reachable |

**Table 5.2:** Room class parameters

Due to the path planning algorithm structure (*node graph - NAV2 navigation*), it could happen that based on the map rooms shape, node graph links and NAV2 computed path may not coincide because NAV2 computes always the shortest path to connect current position and goal, without being aware of the possible presence of doors along the route.
For this reason can be useful to divide an area in two or more Rooms instead of one, using their fictitious and coincident Exit nodes as intermediate destinations of the entire path (MORE OF THIS IN FURTHER IMPROVEMENTS, LA PARTE DOVE SI PARLA DELLA MODIFICA ALLA COSTMAP INVECE DI USARE

| | **Points** | **Exit X.1** | ... | **Exit X.m** |
|---|---|---|---|---|
| $x$ | $[x_0, \ldots, x_p]$ | $[x_0, \ldots, x_{e_1}]$ | ... | $[x_0, \ldots, x_{e_m}]$ |
| $y$ | $[y_0, \ldots, y_p]$ | $[y_0, \ldots, y_{e_1}]$ | ... | $[y_0, \ldots, y_{e_m}]$ |
| $z$ | $[z_0, \ldots, z_p]$ | $[z_0, \ldots, z_{e_1}]$ | ... | $[z_0, \ldots, z_{e_m}]$ |
| $q_0$ | $[q_{0,0}, \ldots, q_{0,p}]$ | $[q_{0,0}, \ldots, q_{0,e_1}]$ | ... | $[q_{0,0}, \ldots, q_{0,e_m}]$ |
| $q_1$ | $[q_{1,0}, \ldots, q_{1,p}]$ | $[q_{1,0}, \ldots, q_{1,e_1}]$ | ... | $[q_{1,0}, \ldots, q_{1,e_m}]$ |
| $q_2$ | $[q_{2,0}, \ldots, q_{2,p}]$ | $[q_{2,0}, \ldots, q_{2,e_1}]$ | ... | $[q_{2,0}, \ldots, q_{2,e_m}]$ |
| $q_3$ | $[q_{3,0}, \ldots, q_{3,p}]$ | $[q_{3,0}, \ldots, q_{3,e_1}]$ | ... | $[q_{3,0}, \ldots, q_{3,e_m}]$ |

**Table 5.3:** RoomX Points and Exits data structure

| | **Points** | **Exit X.1** | ... | **Exit X.m** |
|---|---|---|---|---|
| $x$ | $[x_{br}]$ | [ ] | ... | [ ] |
| $y$ | $[y_{br}]$ | [ ] | ... | [ ] |
| $z$ | $[z_{br}]$ | [ ] | ... | [ ] |
| $q_0$ | $[q_{0,br}]$ | [ ] | ... | [ ] |
| $q_1$ | $[q_{1,br}]$ | [ ] | ... | [ ] |
| $q_2$ | $[q_{2,br}]$ | [ ] | ... | [ ] |
| $q_3$ | $[q_{3,br}]$ | [ ] | ... | [ ] |

**Table 5.4:** Room0 data structure

IL GRAFO). (INSERIRE IMMAGINI DI ESEMPIO)

### 5.2.2 Nodes links

In order to know how Points and Exits nodes are linked in a room and between different rooms, a $n \times n$ matrix has been implemented, where $n$ is the total number of nodes of the graph. Calling this matrix $W$, it is made by the sum of two matrices:

$$W = A + E \tag{5.1}$$

In $A$ matrix, each element $a_{ij}$ indicates the distance between the $i$-th and the $j$-th nodes **belonging to the same Room**, where $i$ is the row index and $j$ the column one. If two nodes are not linked, the correspondent matrix element is equal to 0. Also all the elements on the main diagonal of the matrix are set to 0 because each $a_{ij}$ with $i = j$ is the link from the $i$-th node to itself.
If there is no difference in the direction of travel of the link between one node and another, it means that:

$$a_{ij} = a_{ji}\ ,\ i, j \in \mathbb{N} : i, j \in [0, n],\ i \neq j$$

In other words, with these conditions the matrix becomes a symmetric one.
Then, analyzing more in deep how nodes are linked, the first rule of all nodes says that all nodes of the same Room are linked together (also the Battery recharge node in case of Room0 and Room1), this leads to the fact that Matrix A is split into $m = num\ rooms - 1$ (-1 because Room0 becomes an additional node in Room1) square symmetric sub-matrices $A_i$ ($i \in \mathbb{N} : i \in [1, m]$) and their dimension $n_i$ is given by the sum of Points and Exits of the room (+1 in the case of Room1). These sub-matrices have all the elements on the main diagonal equal to 0.

$$A = \begin{bmatrix} \begin{bmatrix}\mathrm{A}_1\end{bmatrix} & 0^{n_1\times n_2} & \dots & 0^{n_1\times n_{m-1}} & 0^{n_1\times n_m} \\ 0^{n_2\times n_1} & \begin{bmatrix}\mathrm{A}_2\end{bmatrix} & \dots & 0^{n_2\times n_{m-1}} & 0^{n_2\times n_m} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0^{n_{m-1}\times n_1} & 0^{n_{m-1}\times n_2} & \dots & \begin{bmatrix}\mathrm{A}_{m-1}\end{bmatrix} & 0^{n_{m-1}\times n_m} \\ 0^{n_m\times n_1} & 0^{n_m\times n_2} & \dots & 0^{n_m\times n_{m-1}} & \begin{bmatrix}\mathrm{A}_m\end{bmatrix} \end{bmatrix} \tag{5.2}$$

$$A_i = \begin{bmatrix} 0 & \mathrm{a}_{i_{1,2}} & \dots & \mathrm{a}_{i_{1,n_i}} \\ \mathrm{a}_{i_{2,1}} & 0 & \dots & \mathrm{a}_{i_{2,n_1}} \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{a}_{i_{n_i,1}} & \mathrm{a}_{i_{n_i,2}} & \dots & 0 \end{bmatrix} \tag{5.3}$$

The elements of matrix $E$ instead are the distances between the **relative Exits of communicating rooms**. In particular, two generic elements, $e_{ij}$ and $e_{ji}$, symmetrical with respect to the main diagonal of the , is equal to $d_{ij}$ if $i$-th and $j$-th are Exit A.B(k) and Exit B.A(k) nodes (in other words $i$ and $j$ are associated to same exit between two Rooms). Otherwise $e_{ij} = e_{ji} = 0$. It is possible to notice that also this matrix is symmetrical ($E_{i.j} = E^T_{j.i}$) and presents only elements equal to 0 on the main diagonal.

$$E = \begin{bmatrix} 0^{n_1\times n_1} & \begin{bmatrix}\mathrm{E}_{1.2}\end{bmatrix} & \dots & \begin{bmatrix}\mathrm{E}_{1.m-1}\end{bmatrix} & \begin{bmatrix}\mathrm{E}_{1.m}\end{bmatrix} \\ \begin{bmatrix}\mathrm{E}_{2.1}\end{bmatrix} & 0^{n_2\times n_2} & \dots & \begin{bmatrix}\mathrm{E}_{2.m-1}\end{bmatrix} & \begin{bmatrix}\mathrm{E}_{2.m}\end{bmatrix} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \begin{bmatrix}\mathrm{E}_{m-1.1}\end{bmatrix} & \begin{bmatrix}\mathrm{E}_{m-1.2}\end{bmatrix} & \dots & 0^{n_{m-1}\times n_{m-1}} & \begin{bmatrix}\mathrm{E}_{m-1.m}\end{bmatrix} \\ \begin{bmatrix}\mathrm{E}_{m.1}\end{bmatrix} & \begin{bmatrix}\mathrm{E}_{m.2}\end{bmatrix} & \dots & \begin{bmatrix}\mathrm{E}_{m.m-1}\end{bmatrix} & 0^{n_m\times n_m} \end{bmatrix} \tag{5.4}$$

Also matrix $W$, that is the sum of matrices $A$ and $E$, is a symmetric matrix with all 0 on the main diagonal.

In the code, $W$ takes the name *weights*, and is actually provided as input to the path planning algorithm, with the map and Points and Exits poses. In the case of the simulations that will be shown later, distance $d_{ij}$ between two nodes $i$ and $j$, are computed summing their coordinates absolute differences in the map reference frame:

$$d_{ij} = |(x_i - x_j)| + |(y_i - y_j)| \tag{5.5}$$

This is due to the possible presence of walls and other fixed obstacles that prevent the robot from proceeding in a straight line between two points, so the distance between them will not simply be the minimum one. But also the method presented in equation 5.5 does not work in general, because if fore example two nodes are linked by a 'S' shape corridor the distance between them will be composed by the sum of each linear segment (MORE OF THIS IN FURTHER IMPROVEMENTS).

More information are provided by four lists:

- *room_id* : the $i$-th element of this list is the Room in which is contained the $i$-th node.
- *exit_id* : the $i$-th element of this list is the Room that the $i$-th Exit node points to. If the a node is not an Exit, his correspondent value in this list is -1.
- *index* : this list is used to assign an index to a Room points and to eventual multiple Exits between two Rooms, differentiating Points and Exits because Points indexes are listed from 0 to *num points-1*, instead Exits indexes goes from -1 to -*num exits*.
- *links* : the $i$-th element of this list is 1 for Exit nodes that has a door, 2 for Exit nodes that has not a door (it is possible to avoid the checking door procedure with the ToF sensor), and 0 if a node it is not an Exit.

## 5.3 ROS2 Nodes Network and Schedule Node

The *Schedule Node* is the main ROS2 node of the entire network. It is the behaviour tree of the algorithm, where path is computed, and that handles door checking and sanitation procedures. It takes information from the *ToF Publisher Node* to know if a door is open or closed and in this last case recumpute the path. It can also communicate with the *Sanit Action Server*, because the *Sanit Action Client* is the Schedule Node itself, getting data from the $CO_2$ measures and, thanks to them, the sanitation scheduling is computed.
Schedule node is also responsible for all robot movements, in fact it sends the destination coordinates to the NAV2 Navigation Server and gets feedback from it.

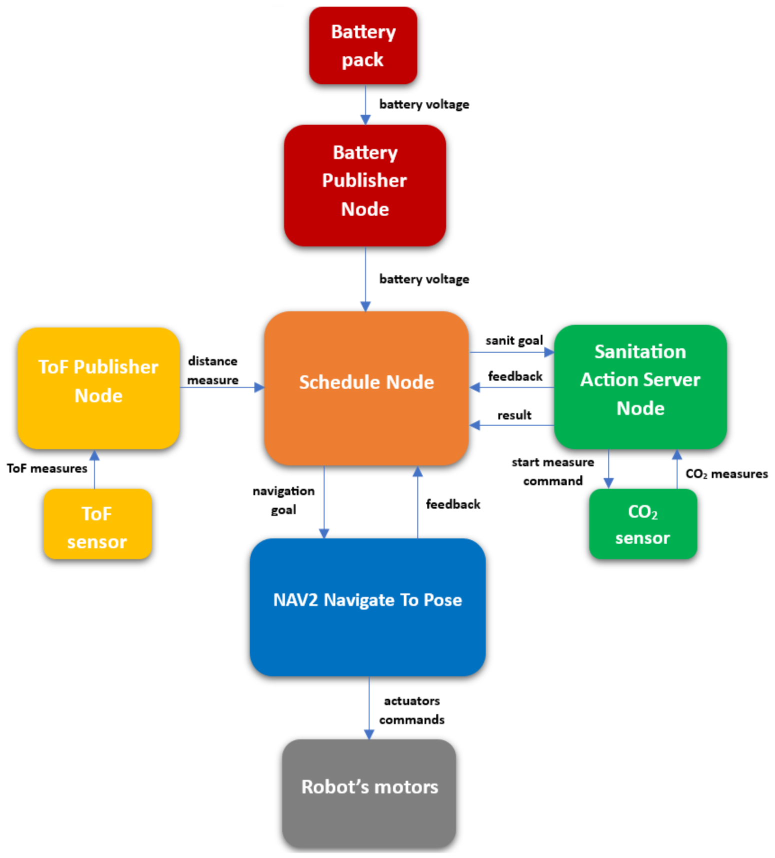

**Figure 5.2:** Nodes network

## 5.4 Sanitation Action Service

The **Sanit Action Server** is the network component responsible for handling the sanitation task. This server communicates with the Arduino Uno board attathed to the $CO_2$ sensor by serial port, and also with the Sanit Action Client that is the Schedule Node, communicating with a custom Action Service, providing to it feedback measurements anf final sanitation results.

### 5.4.1 $CO_2$ sensor data acquisition

The$CO_2$ measurements are performed by the $CO_2$ sensor mounted on an Arduino Uno board that communicates with the Sanit Action Server through serial port. The simplified algorithm of the Arduino code of this sensor is implemented in Algorithm 1.

-if (serial.read() == 10):

**Algorithm 1** Algorithm implemented in the loop section of the Arduino code of the $CO_2$ sensor.

```
if co2_lvl >= 1000 then
    if co2_lvl < 1250 then
        fan_lvl = 15
    else
        if co2_lvl < 1500 then
            fan_lvl = 16
        else
            if co2_lvl >= 1500 then
                fan_lvl = 17 serial.print(-fan_lvl)
while (co2_lvl >=1000):
co2_lvl = co2_reading()
serial.print(co2_lvl)
sanit_need_flag = 1 else: sanit_need_flag = 0
                serial.print(sanit_need_flag)
                $reading \leftarrow serial.read()$
                if $reading$ == "" then
                    procedure ADAM($\alpha, \beta_1, \beta_2, f, \theta_0$)
                        ▷ $\alpha$ is the stepsize
                        ▷ $\beta_1, \beta_2 \in [0, 1)$ are the exponential decay rates for the moment estimates
                        ▷ $f(\theta)$ is the objective function to optimize
                        ▷ $\theta_0$ is the initial vector of parameters which will be optimized
                        ▷ Initialization
                        $m_0 \leftarrow 0$            ▷ First moment estimate vector set to 0
                        $v_0 \leftarrow 0$            ▷ Second moment estimate vector set to 0
                        $t \leftarrow 0$                            ▷ Timestep set to 0
                        ▷ Execution
                        while $\theta_t$ not converged do
                            $t \leftarrow t + 1$                     ▷ Update timestep
                            ▷ Gradients are computed w.r.t the parameters to optimize
                            ▷ using the value of the objective function
                            ▷ at the previous timestep
                            $g_t \leftarrow \nabla_\theta f(\theta_{t-1})$
                            ▷ Update of first-moment and second-moment estimates using
                            ▷ previous value and new gradients, biased
                            $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$
                            $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$
                            ▷ Bias-correction of estimates
                            $\hat{m}_t \leftarrow \frac{m_t}{1 - \beta_1^t}$
                            $\hat{v}_t \leftarrow \frac{v_t}{1 - \beta_2^t}$
                            $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$      ▷ Update parameters
                        end while
                        return $\theta_t$          ▷ Optimized parameters are returned
                    end procedure
```

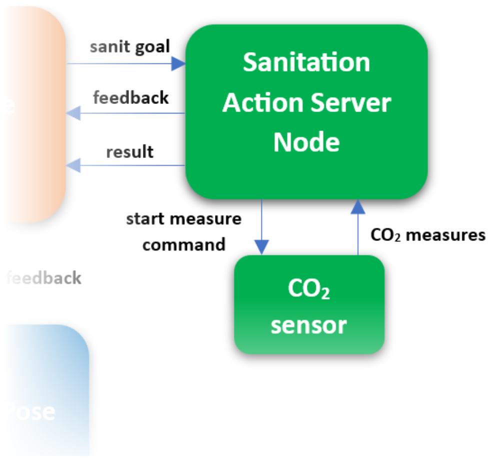

**Figure 5.3:** Sanit Action Server

- co2_lvl = co2_reading()
- serial.print(co2_lvl)
- if co2_lvl >= 1000):
- if (co2_lvl co2 < 1250):
- fan_lvl = 15
- elif (co2_lvl < 1500)
- fan_lvl = 16
- elif co2_lvl >= 1500:
- fan_lvl = 17
- serial.print(-fan_lvl)
- while (co2_lvl >=1000):
- co2_lvl = co2_reading()
- serial.print(co2_lvl)
- sanit_need_flag = 1 -else: - sanit_need_flag = 0 -serial.print(sanit_need_flag)

### 5.4.2 Sanit Action Server and fan control

For the sanitation task a custom ROS2 Action Service was created. This custom Action has the three fields:

- **goal**: *int*32 *start*, is the threshold relative $CO_2$ level in the air in *ppm* (part per million) sent by the Sanit Action Client. When a sanitation process is started, it end when the sensor read a measurement below that threshold. On the current code this threshold is equal to 1000*ppm*, because is the value of $CO_2$ relative quantity in an occupied indoor environment above which air begins to be unpleasant for people and breathing it for prolonged periods can cause problems (FONTE https://www.kane.co.uk/knowledge-centre/what-are-safe-levels-of-co-and-co2-in-rooms).

- **result**: result field contains two *int*32 variables. The first is *sanit_need_flag* (can also be a *bool* variable) is a flag that inform if the sanitation process required the fun intervention (flag equal to 1), or the $CO_2$ level was already under the threshold and sanitation was not required (flag equal to 0). The second is *final_measure*, the final measurement of the $CO_2$, under the goal threshold.
- **feedback**: *int*32 *start*, are the different measures performed by the sensor and send from the server to the client.

From the serial port, the Sanit Action Server node receives also the fan speed commands, that it forwards through the */action_topic*, to the *pic4serial* node, implemented in PIC4SeR, that handles fan commands and other functionalities.

| **first measurement** | **fan speed level** | ***sanit_need_flag*** |
|---|---|---|
| $x <$ threshold 1 | 0 | 0 |
| threshold 1 $\leq x \leq$ threshold 2 | 1 | 1 |
| threshold 2 $< x \leq$ threshold 3 | 2 | 1 |
| $x >$ threshold 3 | 3 | 1 |

**Table 5.5:** Fan control lows

## 5.5 ToF Publisher Node

Information from the ToF sensor are provided by a publisher node called *ToF Publisher*, that communicates with the *Schedule Node* by an *int*64[ ] data type topic */tof_topic*. This node communicate with the sensor through serial port connection with an Arduino board.

### 5.5.1 Data acquisition

For each sensor acquisition, the node takes 64 values one by one from the serial communication. After 64 values the node know that a measurement has ended and start a data processing procedure. The 64 values are inserted in a $8 \times 8$ array matrix. Then in the *ToF Publisher Node* the average of these 64 elements is computed for four successive group of measurements, and for these for avarages is compute again the avarage and the maximum difference between this values. If certain thresholds are respected on the final average and on the maximum difference, the door will be considered opened.

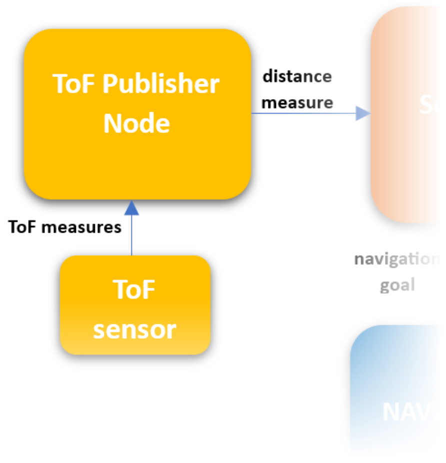

**Figure 5.4:** ToF Publisher Node

### 5.5.2 Exit-Door minimum distance

The task of checking the opening of the doors needs some considerations. Due to NAV2 tolerances in considering a goal pose successfully reached, $0.25m$ for the position part of the goal and $0.25rad$ for the orientation one, it must be computed a maximum distance where the Exit node can be located respect the door position. In any case this distance must be less than $4m$ that is the sensor maximum measurement range.
These tolerances create a triangular tolerance area and to be sure that the ToF measurement range is completely on the door surface, the base of this triangular shape must be contained inside the door horizontal extension. To compute the minimum distance from the Exit node and the corresponding door, we can imagine a reference system with origin in the Exit node, with the $y$ axis contained in the ground plane and the $x$ axis perpendicular to the door plane and contained in the plane, perpendicular to the ground plane, that divide in two halves the door (Figure 5.5).
The intersection between the $x$ axis and the door is the point $M(x_m,0)$, $L$ and $L'$ are the two extremes of the door ($LL'$ is the door width, $LM = ML'$). The origin of the reference frame is also he center $C(0,0)$ of the circumference of radius $R$ equal to the positional tolerance of NAV2. The reference orientation of the Exit node has the same direction of the $x$ axis. The ToF direction during the measurement can be misaligned of an angle $\beta$ equal to angular tolerance of NAV2.

If the purpose is to obtain that the entire measure range of the ToF is projected on the door surface, the worst case is when the robot position (assuming that the ToF

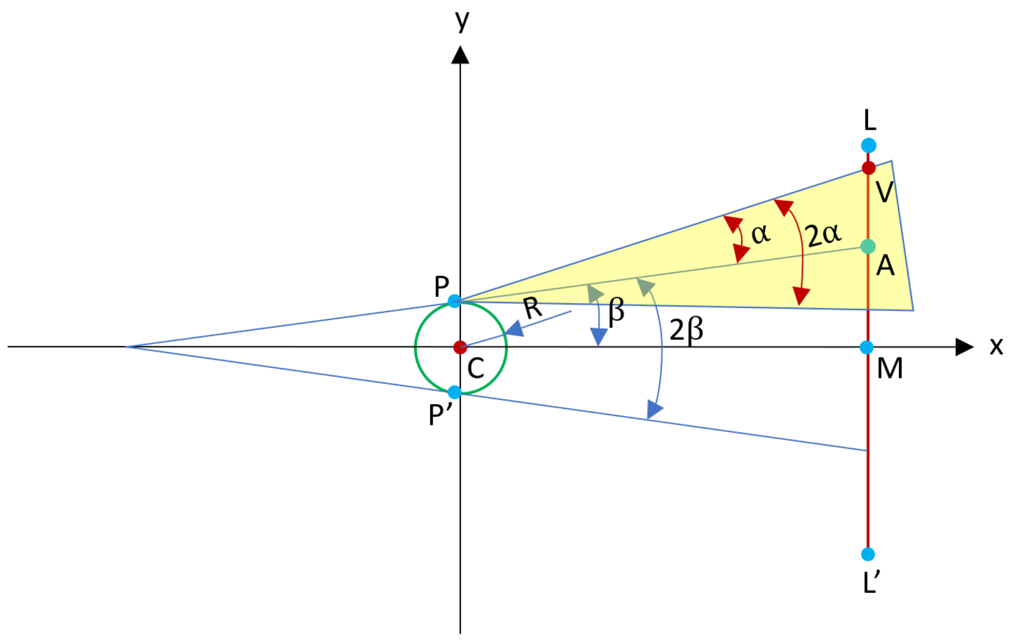

**Figure 5.5:** Exit node - door minimum distance computation

sensor position coincides with the robot center) is on the tolerance circumference and the angular error is the maximum one. The biggest error is obtained when the robot is on the tangent point between the tolerance circumference and the maximum misalignment tolerance direction (circumference of center $C$ and radius $R$ and segment $PA$, where $P$ is the tangent point). From this position the external margin of the ToF sensor measurement has an ulterior angle drift of $\alpha$ that is half the measurement angle range width.
More details about the computations inside the Appendix (RIFERIMENTO ALLA APPENDIX)

## 5.6 Battery Publisher Node

(IMMAGINE ROS NETWORK CON FOCUS SU BATTERY NODE) This node is not been already implemented, but is very similar to the *ToF Publisher Node*, the only difference will be the source for the battery voltage measures that very probably will not be provided through serial port but another source.
Then the Battery Publisher Node will publish the measurements on the */battery_topic*, from where the *Schedule Node* can read them.

# Chapter 6

# Schedule Node algorithm

The *Schedule Node* is the main node of the network. Inside it there are all the fundamental logics for navigation and robot tasks management. It works as a four-states machine that handles all the different situation the robot can faces during his operating time. Inside the Schedule node is declared the Room python class for data structure, and a lot of variables, flags and methods.
The main method, called *main_program_callback* is the responsible for the states switching using a *status flag variable* for each state. These state are *Navigation management state*, *Door management state*, *Sanitation management state* and *Battery management state*. It also provides the destination node coordinates to the NAV2 Navigation Server. For this purpose there are four indices variable:

- *current_room*: is the variable that keeps track of the current room in which the robot is.

- *next_room*: this variable contains the room in which is located the next node in the schedule that the robot must reach.

- *j*: used to choose the sub-list in the parameters lists, so to differentiate the nodes set (Points, Exit X.1, Exit X.2, ..., Exit X.m). $j$ must be 0 if the next node is a Point or, it is a Exit, it must be equal to the Room number of the one that communicates with the current Room X through that door. For example if the next node of the path is the Exit 2.3 and the current Room is Room2, $j$ must be equal to 3.

- *k*: used to choose the node inside the $j$-th sub-list. This variable allows to choose the desired node inside the Points group, or if an Exit has more nodes, also the correct Exit nodes.

When a new destination room is found, a new *sequence* struct is computed. This data struct contains the list of rooms from which the robot need to pass to reach

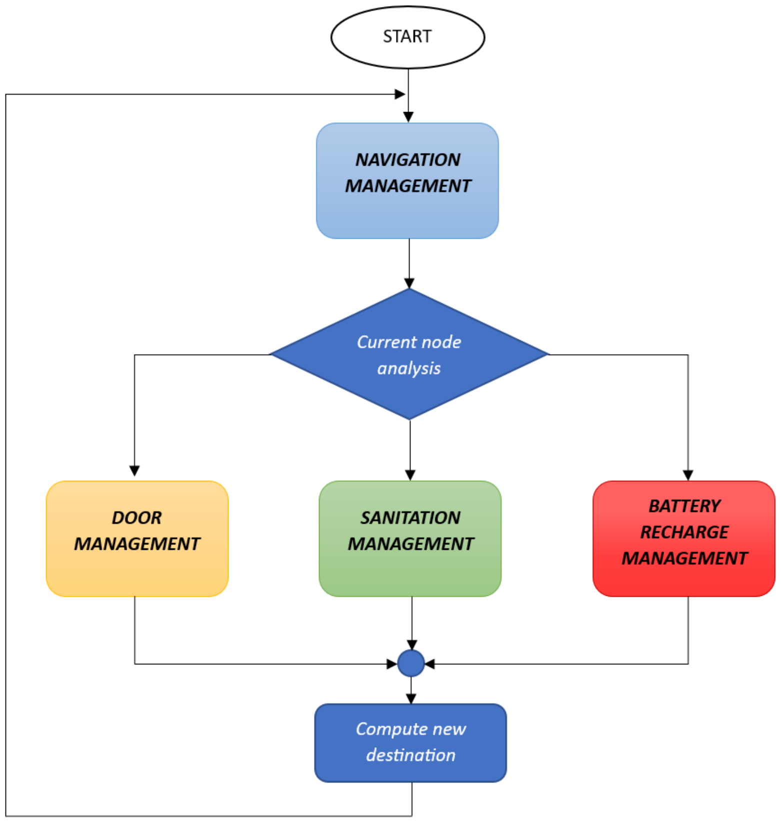

**Figure 6.1:** Schedule node main method scheme

the destination. An example of the *sequence* struct is provided in Table 6.1.
The selection of the temporary destination is made using the *i* index variable,

| *room_id* | *exit_id* | *index* | **node number** |
|---|---|---|---|
| 2 | 0 | 1 | 12 |
| 3 | 2 | 1 | 15 |
| 4 | 3 | 0 | 19 |

**Table 6.1:** Sequence data structure example

that starts from the value *len(sequence)-1* and decrement by 1 every time a mid destination is reached and there are not closed doors along the path. Referring to the example in Table 6.1, the struct is read by the bottom to the top, in this case the first node where the robot must go is the the node 19, the Exit 4.3(0) node. The next one is node 15, Exit 3.2(1) node (the 1 in the parenthesis means that there are more entrances between the two rooms and the algorithm chooses the one with index 1). The last node is the destination of the path, it is the Point(1) in Room2, and once there the robot will start the sanitation procedure. It possible to notice that the first row of *sequence* will always contains a Point (or Battery recharge node), because only Points can be a destination for the path, in fact the *exit_id* of this row will be always 0 because it as a Point, not an Exit. All the others rows contain an Exit nodes of passage rooms.

Each state function block has the same structure divided in two main parts, each part is triggered by a flag (*flag1* for the first part, and *flag2* for the second one):

1. this part of the block set *flag1* equal to 1 in order to prevent the continuation of the execution of *main_program_callback*, allowing the execution of the associated callback. This part also sets the *flag2* to 0, so when the *main_program_callback* will be executed again, a repetition of the callback function is avoided until it is needed again.
   Once the callback is executed, after its functional part, it sets again to 0 the *flag1*, so when *main_program_callback* is performed another time, the algorithm proceeds with the next part of the functional block.

2. if *flag1* is equal to 0, the second part of the block is executed. Inside this part, tacking as input the results provided by the callback (information about the door opening for the *Door management state*, or if the navigation is successfully ended or not for the *Navigation management state*, and so on...), the functional block performs its operations. At the end of this part, *flag2* is again initialized to 1 and *flag1* to 0, allowing the functional block to execute the callback again.

At the end of the *main_program_callback*, if everything worked as intended and no critical navigation situation arises, there the computation of indices variables new values that has to communicated to NAV2 to follow the computed path.

## 6.1 Navigation phase

The tasks of this functional block are sending the pose coordinates of the destination node to NAV2 and monitoring feedback from the Navigation Server in order to understand if the navigation succeeded or failed. *flag1* and *flag2* of this block are *in_nav_flag* and *task_compl_flag*. *task_compl_flag* is set again to 1 when the *main_program_callback* reaches its end. This is due to the fact that a navigation should be performed only when the robot task on the current node is ended.

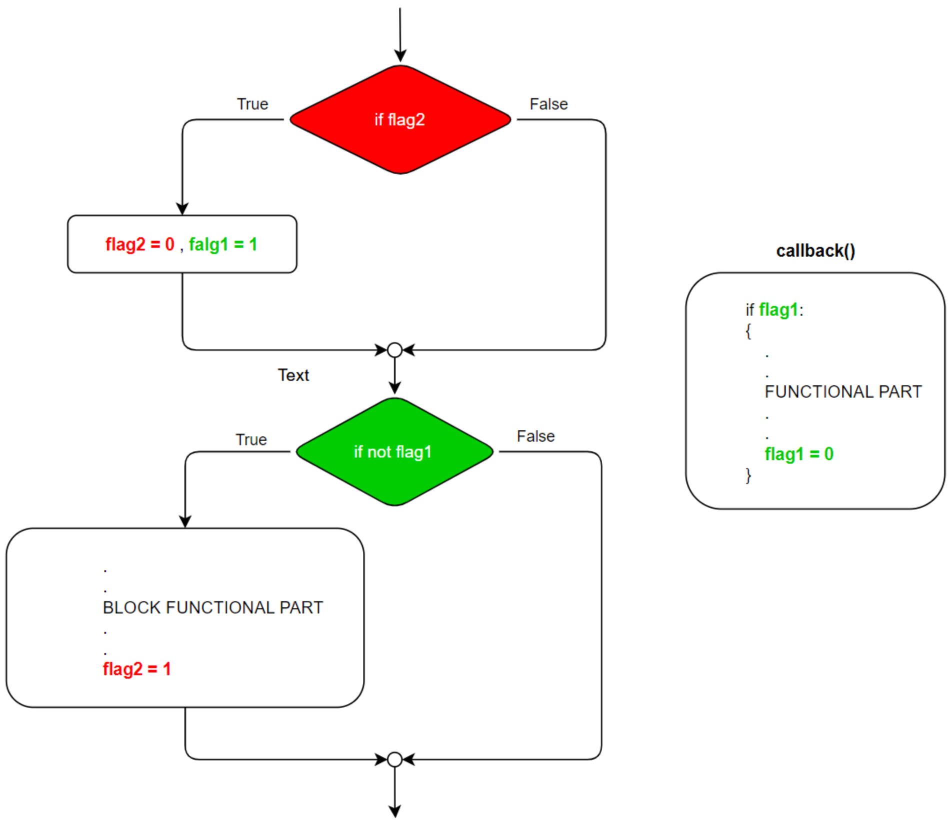

**Figure 6.2:** Functional block scheme example

At the end of the navigation phase, there is a dedicated section for the updating of the room variables *current_room* and *next_room*.

### 6.1.1 Methods and Callbacks

The method responsible for sending the coordinates is *reach_a_pose*. Inside this function the *coord* variable is defined, of type *PoseStamped*, and its parameters $x$, $y$, and the others are filled with the command coord.x = rooms[next_room].x[j][k], coord.y = rooms[next_room].y[j][k], ..., then the *coord* avriable is sent as goal to NAV2. After this, the two flags are setted as in Figure 6.3.

Then, when the *in_nav_flag* is on, the *nav_feedback_callback*, triggered by the */odom* topic, checks from the *robot_navigator* method (deveolped in PIC4SeR) the navigation status (succedeed, failed, unknown) provided from the NAV2 Navigation Server. Every time that navigation fails, the goal is sent another time, until

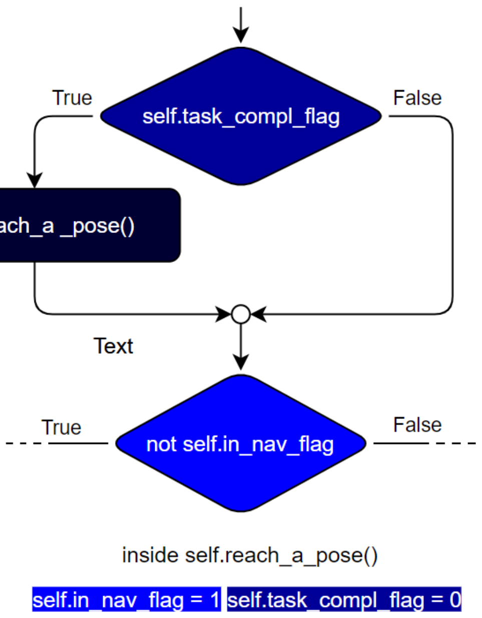

**Figure 6.3:** Navigation phase

the nevigation result is "succedeed".

It could happen that a goal is aborted by an intrinsic error of NAV2 and it is impossible to go on with navigation. For this reason another timer callback has been created, *nav_timeout_callback*, that every time it is executed, is *in_nav_flag* is on, increments a counter. When the counter reaches its maximum value the callback cancels the current goal and sends a new copy of the goal.

## 6.2 Door opening check

The *Door opening check* section of the algorithm has the task to execute the *tof_callback* in order to get the ToF sensor measure and understand if the door in front of the robot is open or not. To enter this section of the *main_program_callback* the following condition must be respected: $j > 0$, that means the robot is actually on an Exit node. Also the this block has the usual structure, but in this case, the first part of the block could be skipped if the connection between the two Exit nodes is without a door (Figure 6.4), because in that case there is no need to measure a distance.

After retrieving the information of the door opening, there are two possibilities:

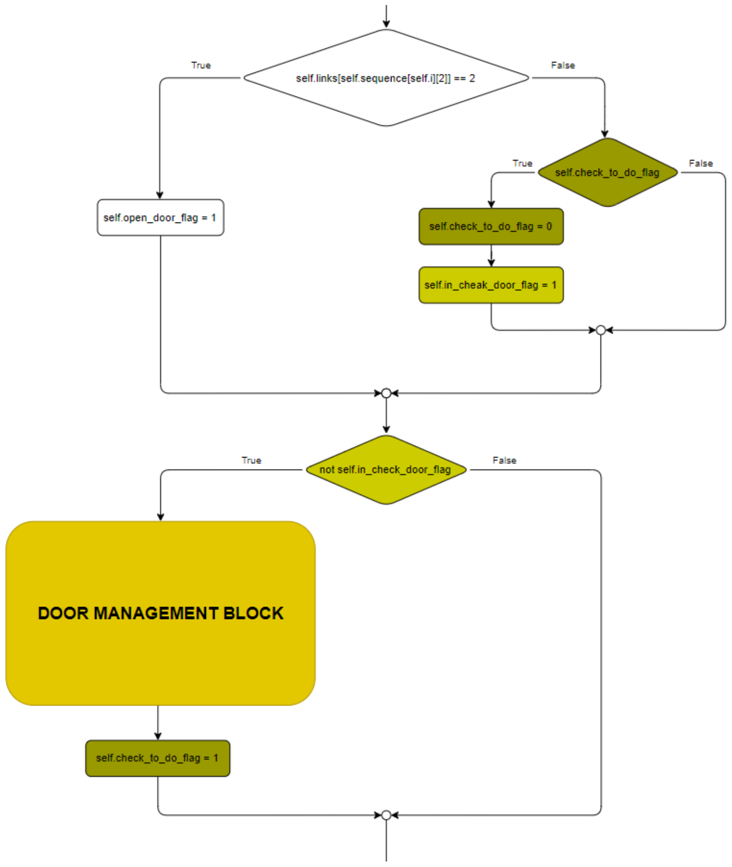
self.links[self.sequence[self.i][2]] == 2
True
False
self.check_to_do_flag
True
False
self.open_door_flag = 1
self.check_to_do_flag = 0
self.in_cheak_door_flag = 1
not self.in_check_door_flag
True
False
DOOR MANAGEMENT BLOCK
self.check_to_do_flag = 1

**Figure 6.4:** Door opening check phase

- **door is closed**: the current path has cannot be followed so it is necessary to recompute the path to the destination room. In this case, in a copy of the *weights* matrix, *updated_weights*, used for Dijkstra algorithm computations (VEDERE SOTTOCAPITOLO DEL new_next_dest_meth), the $w_{ij}$ and $w_{ji}$ elements ($i$ and $j$ are the nodes of the relative closed door) are substituted with 0 interrupting the connection between the two nodes, so the algorithm can recompute the path knowing that in that point the nodes network is interrupted.

- **door is open**: the navigation continues normally, the $i$ index of the *sequence* struct is decremented by 1 and the $i$-th node of the sequence is passed as new destination to NAV2. Also in this case could be necessary to recompute the path because if the battery needs to be recharged (*battery SOC* $<$ *threshold*), but without updating links on the *updated_weights* matrix.

After a while, substituted link must be set to the original value contained in *weights*, otherwise the robot will never pass again from that door. To do this another timer callback was implemented. The *weights_update_callback* uses a *weights_counter* matrix, that has the same dimension of *weights* and *updated_weights*, but inside have all zeros. When a link is canceled in *updated_weights*, the correspondent element in *weights_counter* is set to a max counter number. Every time the *weights_update_callback* is executed, counters different from 0 are decremented until they return to 0. When this happen, the correspondent elements of *updated_weights* are restored to the original values of the *weights* matrix and algorithm can use that link again for the path computation. (IMMAGINE MATRICI weight update e counter)

## 6.3 Rooms sanitation

This section of the algorithm has to handle all sanitation tasks: $CO_2$ measurements and eventual fan activation for sanitation. Also this block is implemented with the same structure of the others, even if is not strictly requested because the Action Service communication already has a protocol where client and server wait for the responds without going on with the script.

### 6.3.1 Sanitation process

. When the robot arrive on the node indicated in the first row of *sequence* struct, $current\_room > 0$ and $j = 0$ conditions are respected, so the algorithm knows that a sanitation procedure has to start (the first condition is needed to differentiate the sanitation procedure to the battery recharge one, performed in Room0). For the sanitation task, the following Room class parameters and variables are needed:

- Room parameter *points_sanit_flag*: a list of dimension equal to the Room *num_points*. Each list element represents the sanitation condition of the respective Point. When the robot is on the Point node $a$, after the sanitation process, the correspondent $points\_sanit\_falg[a]$ is set to 1. If the *sanit_return_code* of the Sanit Action Client is equal to 1, the *points_sanit_falg* of the next Points in the order of sanitation is set to 0 so in that node, sanitation process must be executed another time if not did already.

- Room parameter *points_order*: this list has the same dimension of textit-points_sanit_falg of the same Room. In this list is saved the sanitation order of the Points, in order to know when a new sanitation cycle is starting.

- variable $w$: $w$ is the index that starts from 0 every time a sanitation process begins and is incremented by 1 after each node sanitation, until it reaches the Room *num_points* value and restarts from 0. When during the sanitation, the robot goes on a Point node $a$ for the first time, the *points_order*$[a]$ becomes equal to $w$.

- Room parameter *sanit_tries_counter*: if the room sanitation does not ends when $w$ becomes again 0 (the sanitation process is restarting from the first Point), the *sanit_tries_counter* of the room is incremented by 1.
- Room parameter *attempts*: when the *sanit_tries_counter* reaches the *max_sanit_tries* of the Room, the sanitation is aborted and the *attempts* parameter of the Room is increased by 1. The *attempts* value is increased also when the Room is selected as destination but it is not reachable.

A Room sanitation is definitely ended when all the *points_sanit_flag* of the Room are equal to 1 (success case), ore when Room *attempts* reaches the *max_attempts* value (failure case).

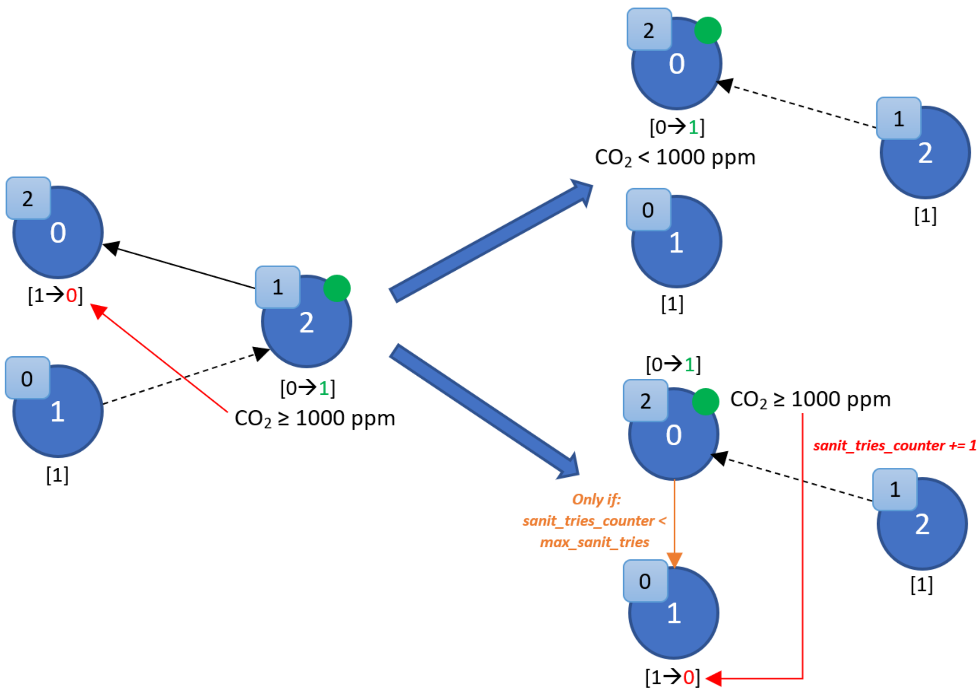

**Figure 6.5:** Sanitation procedure example

A sanitation process example is shown in Figure 6.5: a Room with three Points is in the sanitation process. The numbers inside the circular blocks are the *Points indexes*, numbers inside square blocks are the *points_order*, and in the square brackets are represented the *points_sanit_flag*.

The list of variables and parameters values at the starting time instant is the following:

- $num\_poits = 3$
- $points\_sanit\_flag$ = [1,0,1]
- $points\_order$ = [2,0,1]
- $sanit\_tries\_counter = 1$
- $w = 1$

The robot is on Point 1, and had already completed the sanitation process. Then the robot select as next destination Point 2 (the nearest Point with sanit flag equal to 0).
The actual situation is the one represented in the left part of the image. The $CO_2$ measurement is performed and an air sanitation is required.
Now $points\_sanit\_flag$ = [1,1,0] and $w$ becomes 2.

Robot then move in Point 0, the last Point in the sanification order. This scenario can evolve into three situations, the first represented in the top right corner of the image, the other two in the bottom right one:

**1)** $CO_2$ measurement is below 1000 ppm, so $points\_sanit\_flag$ become [1,1,1], all the sanit flags are equal to 1 and the sanitation ends with success.

**2)** $CO_2$ measurement is grater or equal to 1000 ppm, $points\_sanit\_flag$ become [0,1,1], $w = 3$ that is equal to $num\_points$ so it restart from 0 and of consequence $sanit\_tries\_counter = 2$:

  **2.a)** if $sanit\_tries\_counter = max\_sanit\_tries$, the sanitation process is aborted.

  **2.b)** if $sanit\_tries\_counter < max\_sanit\_tries$, a new sanitation cycle starts.

## 6.4 Battery recharging

When a sanitation schedule is ended or when the battery must to be recharged, the robot destination is the Battery Recharge Node in Room0. Th functional block that handles this situation is the *Battery recharging* phase.
To enter this state functional block, $current\_room$ and $j$ must both be equal to 0.

## 6.5 Next destination method

## 6.6 Next point method

## 6.7 Compute new path method

# Chapter 7

# Simulations and results

## 7.1 CO$_2$ sensor and fun simulation

## 7.2 ToF sensor simulation

## 7.3 Battery simulation

## 7.4 SEGNALI bag file e plot$_bag.py$

## 7.5 Simulations

### 7.5.1 Robot-only simulations

#### Simulation 1

#### Simulation 2

#### Simulation 3

#### Simulation 4

#### Simulation 5

#### Simulation 6

#### Simulation 7

### 7.5.2 Simulation with people

Chapter 8

# Conclusions and further improvements

- SENSORI E CAMERE ALL'INTERNO DELLE STANZE –> MIGLIORE PLANNING
- Con monitoraggio interno alla stanza si può anche settare un diverso livello di priorità in base al numero di persone all'interno, e alle loro azioni: tosse, starnuti, ecc...
- SENSORI SULLE PORTE (PORTE IN VETRO –> DA CAMERA è DIFFICILE CAPIRE SE SONO APERTE O CHIUSE)
- Non bisognerebbe più modificare il grafo (o la costmap) temporaneamente, ma si sa esattamente quando la porta è aperta o chiusa
- NUOVA SANIFICA A TEMPO INVECE CHE MONITORANDO IL LIVELLO DI CO2 (CAUSA CHE QUEL PURIFICATORE NON RIDUCE CO2)
- RETE NEURALE PER ACQUISIZIONE STANZE E RILEVAMENTO AUTOMATICO DEI POINTS PER LA SANIFICA
- LUNGHEZZE DEI LINK DEL GRAFO CALCOLATE AUTOMATICAMENTE CON ComputePathToFollow
- ABBANDONO DEL GRAFO E MODIFICA COSTMAP
- COLLEZIONAMENTO DI DATI PER CAMBIARE LA PRIORITà E ALTRI PARAMETRI DELLE STANZE IN BASE AI DATI RACCOLTI. ESEMPI: se una stanza è quasi sempre chiusa o inutilizzata (con monitoraggio interno) la sua priorità si abbasserà rispetto alle altre. Oppure se una stanza raramente

si riesce a sanificare a prima botta si possono aumentare man mano i suoi cicli di sanifica e i tentativi massimi di sanifica, o se una stanza è ha dei livelli di CO2 quasi sempre alti o ci sono spesso molte persone all'inerno (monitoraggio interno) la sua priorità sarà più alta.

- IMPLEMENTARE CHE SE BATTERIA SCARICA, PER CAMBIARE DESTINAZIONE NON BISOGNA ASPETTARE DI ARRIVARE AL PROSSIMO GOAL MA SI INTERROMPE LA NAVIGAZIONE IN QUALSIASI MOMENTO E SI RICALCOLA PER ANDARE A RICARICARE
- STIME CONSUMO BATTERIA IN FUNZIONE DI f(USO MOTORI, USO VENTILATORE) (CHE POI DIVENTANO DUE FUNZIONI SEPARATE f(MOTORI) E f(BATTERIA), PERCHè MOVIMENTO E SANIFICA NON AVVENGONO MAI INSIEME) PER POTER IMPLEMENTARE UNA GESTIONE DELLA BATTERIA INTELLIGENTE, CHE IN BASE AL LIVELLO VEDE QUALI STANZE PUò ANDARE A SANIFICARE E QUALI NO.
- MODIFICA DI DIJKSTRA IN MODO CHE A PARITÀ DI DISTANZA DA PERCORRERE VENGA SCELTO IL PATH CON IL MINOR NUMERO DI ACCESSI CON PORTA

# Bibliography

[1] stockfeel. «What Is a Service Robot? Practice the vision of intelligent service application». In: *MARKET PROSPECTS* (2022). URL: `https://www.market-prospects.com/articles/what-is-a-service-robot/` (cit. on p. 1).

[2] Piero Formica. «Human Manufacturing, l'unisono di Comau che fa parlare agli operatori la lingua dell'automazione». In: *INDUSTRIA ITALIANA* (2020). URL: `https://www.industriaitaliana.it/comau-robotica-automazione-human-manufacturing-horizon-2020-fca/` (cit. on p. 2).

[3] *ROS - Robot Operating System*. ROS.org. URL: `https://www.ros.org/` (cit. on p. 9).