UNIVERSITÀ DI BOLOGNA

School of Engineering

Master Degree in Automation Engineering

Autonomous And Mobile Robotics

SANITIZATION PROJECT REPORT

Professor: **Gianluca Palli**
Tutor: **Kevin Galassi**

Students:
Francesconi Nicola
Mengozzi Riccardo
Vespignani Lorenzo

Academic year 2023/2024

Contents

Introduction

The purpose of this project is to setup a simulation on Robot Operating System 2 (**ROS2**) that uses for simplicity a Turtlebot3 Burger representing a robot for environment Sanitization. To achieve this, after learning the basics of ROS2, we were required to:

- **Setup the environment**: Prepare the simulation on Gazebo Simulator
- **Autonomous Exploration**: Develop an algorithm to let the robot autonomously explore and map an unknown environment
- **Localization and Navigation**: Given a map of the environment the robot must be able to localize itself in this map. After the localization has been completed, the robot must be able to read a set of goals from a text file and navigate to them.
- **Sanitization of Rooms**: Whenever a room is requested to be cleaned, the robot must reach that room, whatever its initial position, and then sanitise it.

(a) TurtleBot3 Burger

(b) Example of sanitizer we want to imitate

1 Task 1 - Environment Setup

This project has been developed and tested using the Turtlebot Burger, figure 1a, which has to explore, map and sanitise the Big House Scenario, figure 2.

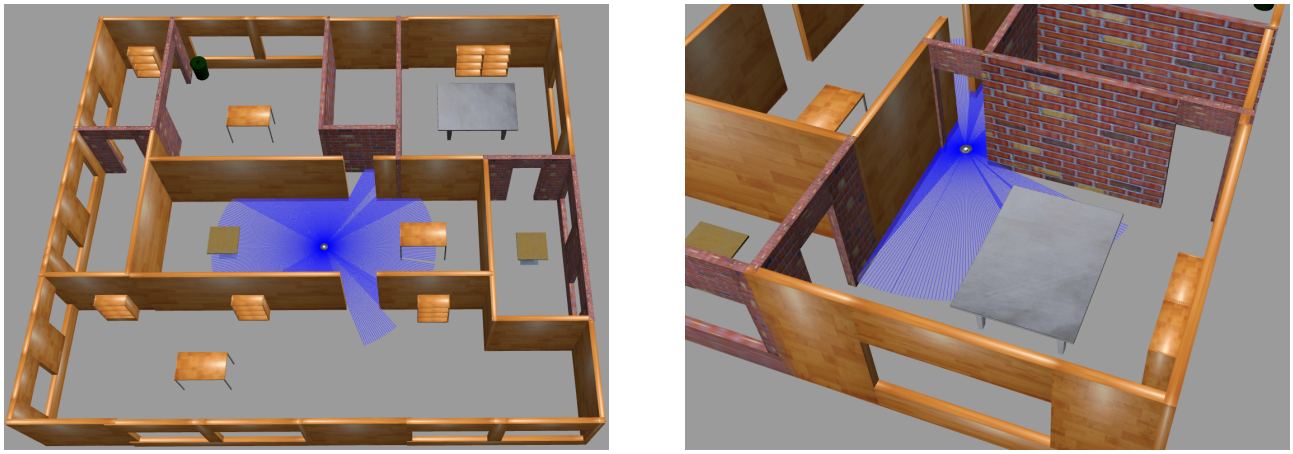


Figure 2: Big House Gazebo environment

Pre-existing packages such as **turtlebot3_navigation2**, **turtlebot3_gazebo** and **turtlebot3_cartographer** were used in this project. Additional custom packages were created by us, in particular a package for autonomous exploration of unkwown environment and a package for sanitization. In our workspace the visible packages are:

- **Turtlebot3_Gazebo** : Contains all the files necessary to run the world with all it's elements.
- **Exploration**: Contains the node file that manages the autonomous exploration task.
- **Sanitization**: Contains all the necessary files for visualization on rviz and the necessary nodes to achieve the sanitization task:
  - **Localization**: Node necessary for localize the robot in a known environment.
  - **Planner**: This node define a set of goals necessary to sanitise a room.
  - **Navigation**: Node used to read the set of goals of the planner and navigate through them.
  - **Sanitization**: This node is the one used for the update of the energy through visible cell whenever the lamp is active.

2 Task 2 - Autonomous Exploration

2.1 Introduction

The goal of the task is the mapping of an unknown environment with the use of Turtlebot3 Lidar.

2.2 Position of the Robot

Firstly, it is important to be able to know the position of the robot inside the generated map. This is done reading data from the `/tf` topic and then apply the proper transformation to compute the position of the base link of the robot with respect to the origin of the map.

2.3 Frontier-Based Exploration

The most important part of the task is to understand which goal to give to Turtlebot. To do that, data from the generated map is used. Here, every pixel has a value between 0 and 100 and is marked as:

- Free/Completely known space: value = 0
- Partially known space: value =]0,100[
- Obstacle: value = 100
- Unknown space: value = -1

This data has been used to set up a **Frontier-Based approach** to explore the map. A frontier is the intersection between free space and unknown space in the map. These regions are good to define goals because they lead the robot to unknown space gradually: Turtlebot can reach them easily and there is less risk for it to bump into obstacles.

In this case, dividing the map in tiles of 5x5 pixels, the frontiers regions have been defined as those tiles having more than 40/50% of pixels with values between 15 and 60 (partially known pixels). The highest the percentage, the highest the priority of the tile. Also, to prevent the robot to move back and forth in the map, a weight depending on the distance has been added: the closer the tile is to a predefined distance value (1-2 meters), the higher its priority will be. So, in the end, the expression of a tile priority has been expressed as:

$$\mathbf{P} = \mathbf{e}^{-(\mathbf{d}^* - \mathbf{d})^2} \cdot \mathbf{F} \tag{1}$$

where:

$\mathbf{P}$ = tile priority
$\mathbf{d}^*$ = tile ideal distance
$\mathbf{d}$ = tile actual distance
$\mathbf{F}$ = tile frontier percentage

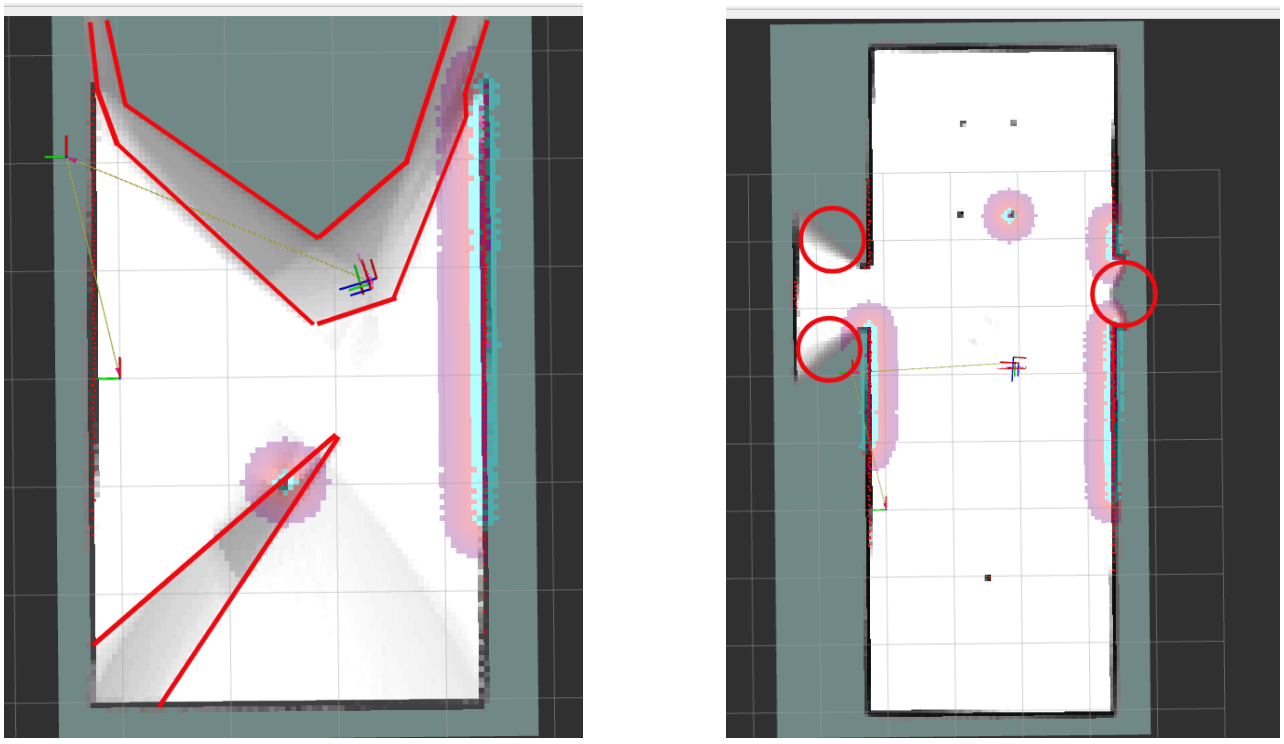


Figure 3: Examples of frontier regions

2.3.1 Wall Map

It can happen that the highest priority goals are on the opposite side of a wall with respect to the robot. Most of the time this is not good, as the ideal strategy would be to finish the exploration of a room before changing it. Other times the choice is forced as the room has been already fully explored, but setting the goal on the other side of a wall could be difficult for the Navigation Stack to handle. To be aware of the positions of all the obstacles with respect to the goals and the robot, a **Wall Map** has been created exploiting the map data. In particular, a tile is said to be an obstacle/wall if at least one pixel inside it has a value higher than 70. This is the result:

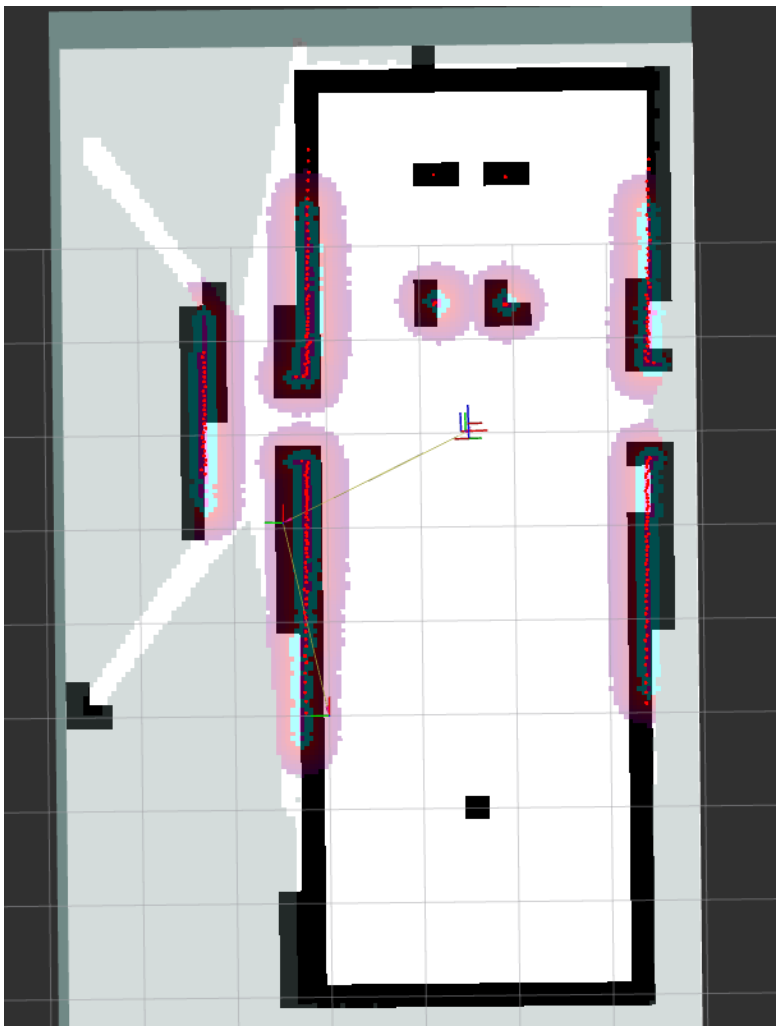


Figure 4: Visualization of the wall map

2.3.2 Bresenham's Line Algorithm

With the wall map, it is now possible to know if an obstacle is present between the robot and a candidate goal, if it is, the goal is discarded. To do that, the line between the goal and the robot is computed, and the search for any obstacle-tile is done with a **Bresenham's Line Algorithm**.

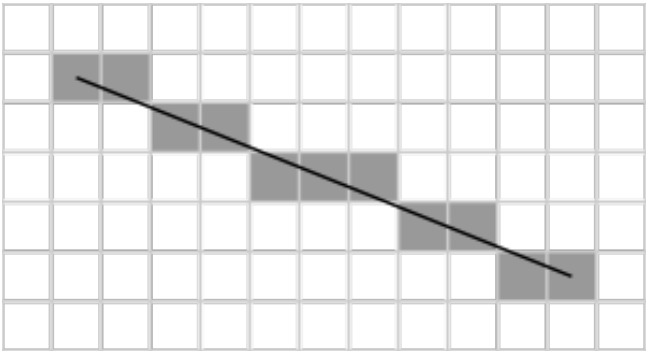


Figure 5: Bresenham's line

2.3.3 Robot-Goal Common Visibility

As it has been said before, sometimes the program is forced to set a goal behind a wall, otherwise the exploration could not go forward. To be sure that the goal would be easily reached by the robot, only certain goals can be chosen. In particular, there must be a point in the map that is linked to both the candidate goal and the robot with a line that doesn't intersect any obstacle, even if there is an obstacle between the robot and the goal.

To do that, the visibility of the robot is computed: all the tiles of the map that do not have obstacles in between with the robot are set to True. Then, also the goal visibility map is computed, starting from the ones with highest priority. Common visibility tiles can now be known simply using a logical AND. If a common visibility tile exists, then the goal can be easily reached by the robot after the common visibility tile is reached. If multiple common visibility tiles exist, then the closest one to the goal is selected.

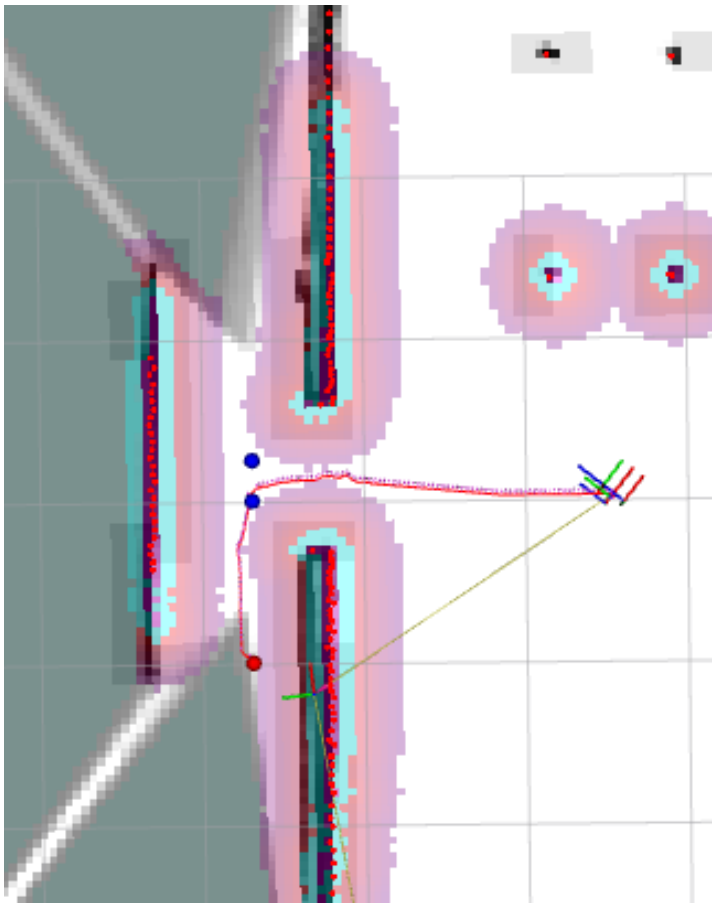


Figure 6: Visualization of common visibility points (blue) and goal point(red)

2.4 Backtracking

Sometimes, when the robot finishes to explore rooms not connected anymore with unexplored regions of the map, it can get stuck, as no more possible goals can be selected. In this case a **Backtracking technique** is used.

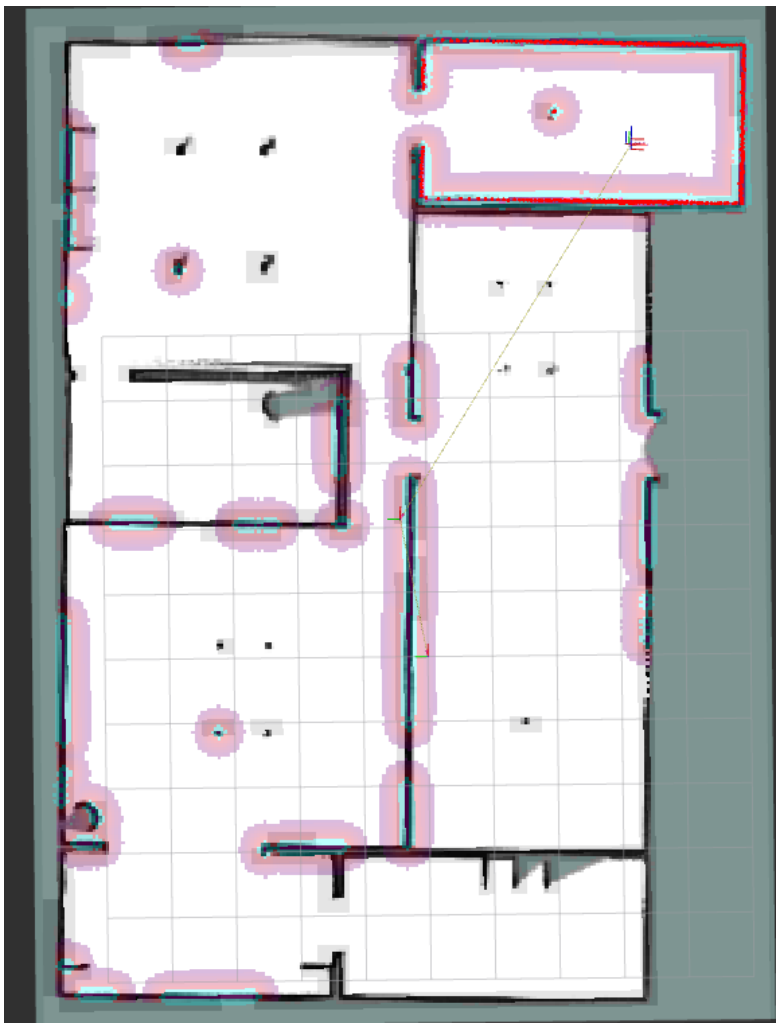


Figure 7: Common situation in which backtracking is needed

Every time a goal is found by the program, it is saved in a goal history array. When the robot get stuck, it searches the closest goal in the history to its present position and it reaches it, canceling all the more recent goals in the history, then it starts to reach all the previous chronological goals until a new goal is found. With this technique the robot should be able to cover its track leaving the room in which it got stuck and find an unexplored region.

3 Task 3 - Localization and Navigation

3.1 Introduction

The goal of this task is to generate a node to perform autonomous localization of the robot in a known environment and a node that given a sequence of goals written in the text file is able to navigate through this goals.

3.2 Localization Node

This node uses the AMCL to localize the robot in the map. AMCL is a localization algorithm used in robotics to estimate the pose (position and orientation) of a robot within an environment. It is a probabilistic approach based on the Monte Carlo method, also known as particle filtering. The algorithm uses a set of particles to represent possible locations of the robot in the environment and updates their weights based on sensor measurements and motion predictions. Over time, as the robot moves and gathers more sensor data, the particles converge to provide a more accurate estimate of the robot's pose.

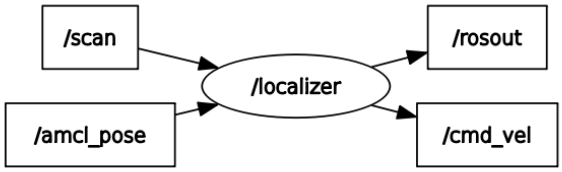
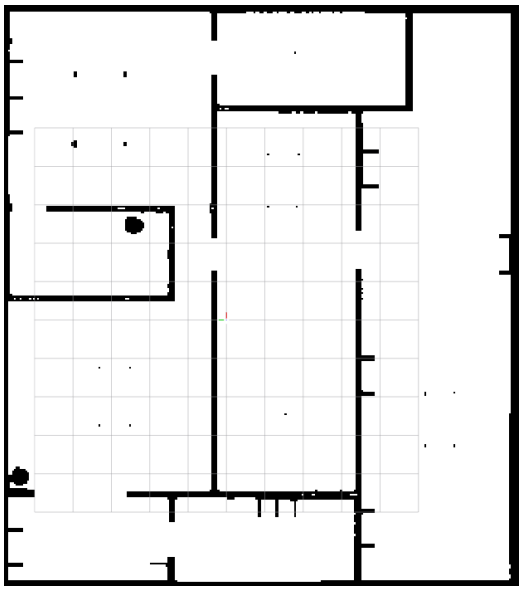


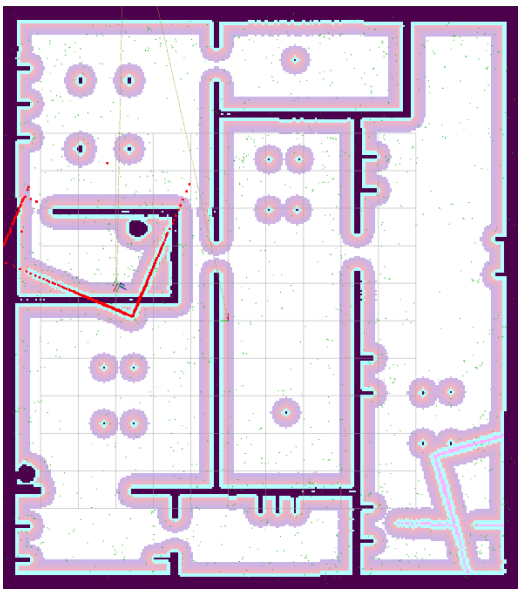
Figure 8: Localization node

The algorithm is quite simple, and is divided in 2 steps:

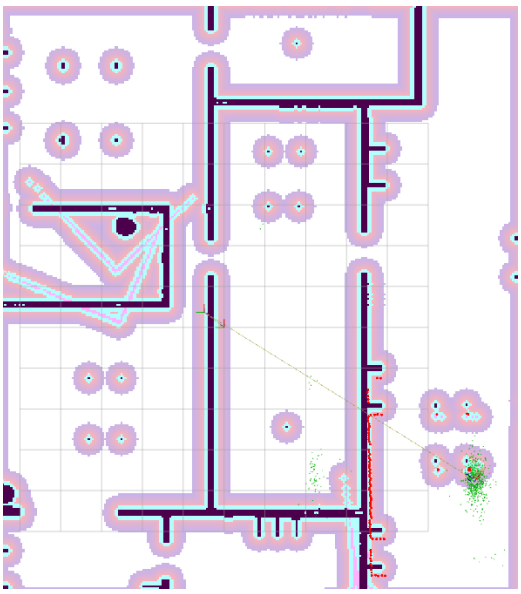
- **AMCL Reinitialization**: At the beginning of the node, the AMCL is reinitialized, so the particles are spwaned in a random way all over the map. From now on the AMCL particles will be updated continuously up to convergence.
- **State Machine Navigation**: This block is a simple "random" navigator that includes also obstacle avoidance. At each iteration of the algorithm, while the sum of the covariance of the AMCL particles is lower than an arbitrary threshold, the node by reading data from the scan topic decides which of the 3 possible action is the best one:
  - **Go Straight**: If no obstacles are detected
  - **Turn Left**: If obstacles are detected on the right of the robot
  - **Turn Right**: If obstacles are detected on the left of the robot



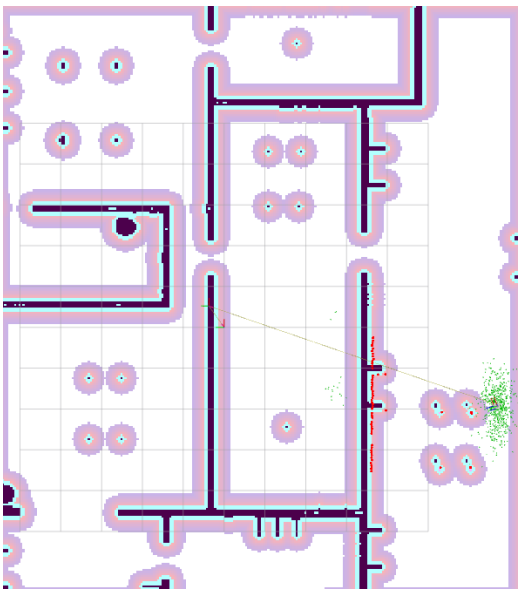
(a) Map Before AMCL Initialization



(b) Global AMCL particle Initialization



(c) AMCL particle Intermediate



(d) AMCL particle Convergence

Whenever the sum of covariance of the AMCL Pose topic is lower than the defined threshold the node clear the global costmap and destroy itself.

3.3 Navigation Node

The purpose of this node is to read goals from a text file and ono at a time achieve these goals. This node basically open a text file where the goal are written in rows as:

Row 1: x_value y_value θ_value
Row 2: x_value y_value θ_value
...

The node append each goal in a list, then through a **BasicNavigator()** the goals are published in sequence and deleted from the list of goals. This node will be useful in the next task. Few modification will be adopted in order to behave properly in the Sanitization task.

4 Task 4 - Sanification of Rooms

4.1 Introduction

The goal of this task is to sanitase some rooms of the mapped environment. In order to do that the previous information, as the map of the environment and the navigation node will be used.

4.2 Room Division

A room division has been done, each room for simplicity has been represented as a rectangular area inside the map as follow:

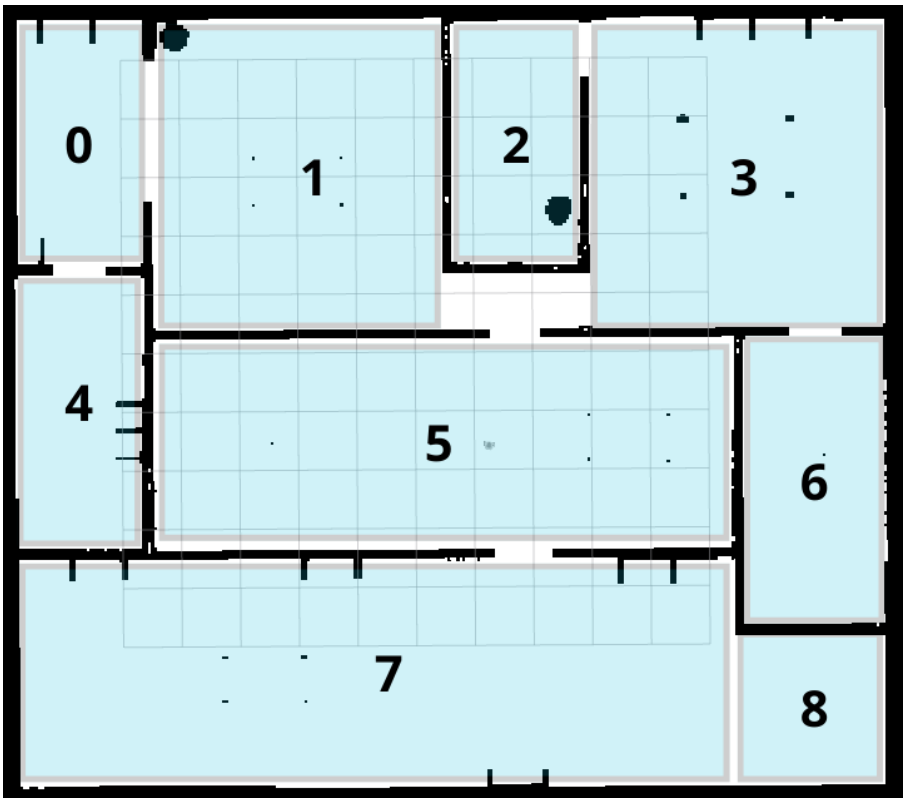


Figure 10: Room Division

Once the room has been defined, the following nodes were implemented in order to accomplish the task:

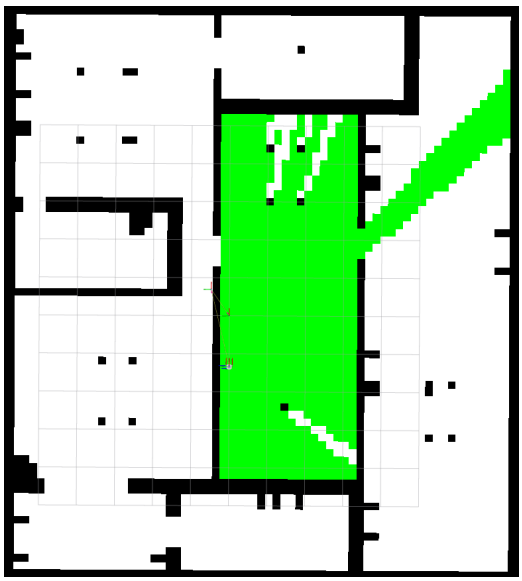
- **Sanitization**: This node is responsible to update the energy over the map when the UV lamp is on only to visible cells.
- **Planner**: This navigate the robot to the room which must be sanitised and plan a sequence of goals necessary for the Sanitization. This goals will be written in a text file that the navigation node must read and reach.
- **Navigation**: This is slightly modified copy of the one discussed in section 3

4.3 Sanitization Node

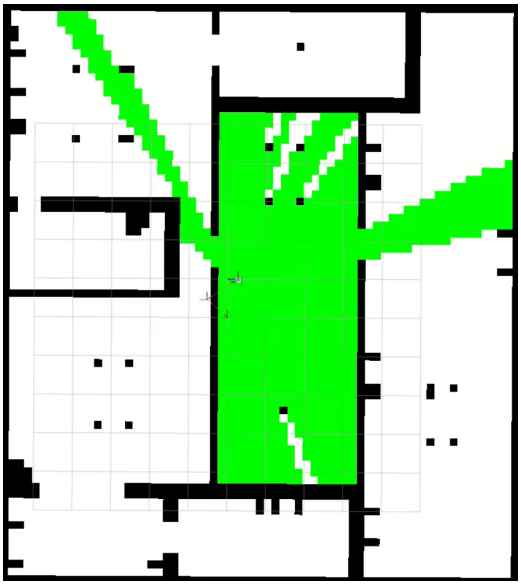
The Sanitization node aims to correctly update the energy of the visible cells of the map. To do that it's important to know the robot position with respect to the map and the obstacle position with respect to the map.

4.3.1 Visibility Update

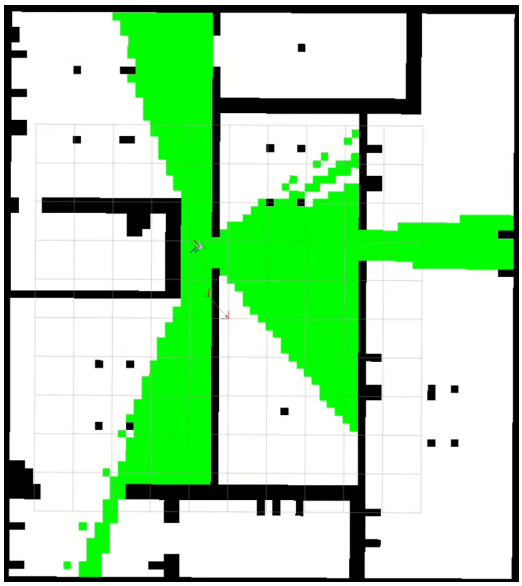
First of all it's important to define which cell are "reachable" from the point of wiew of the robot position. Each cell of the map is marked as: **Obstacle In Between = True / False**, the decision is done by using the same algorithm discussed in section 2 "Bresenham's Line Algorithm" which is able to see if a line pointing to a cell meets an obstacle of the " Wall Map", which is the same map discussed in section 2. Each time the energy must be updated the matrix that tells us which cells are "visible" from the point of view of the robot is updated.



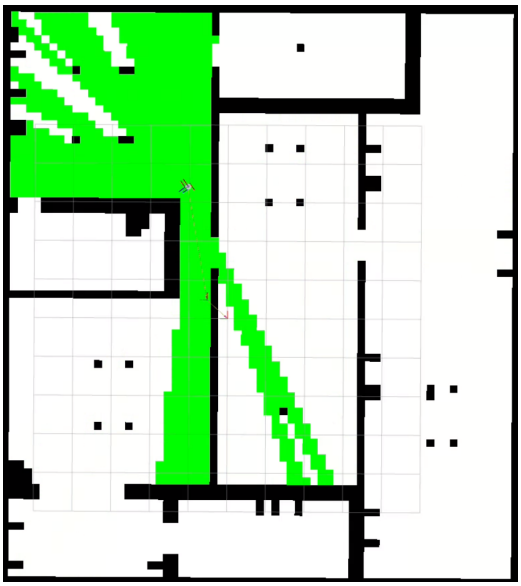
(a) Visibility



(b) Visibility



(c) Visibility



(d) Visibility

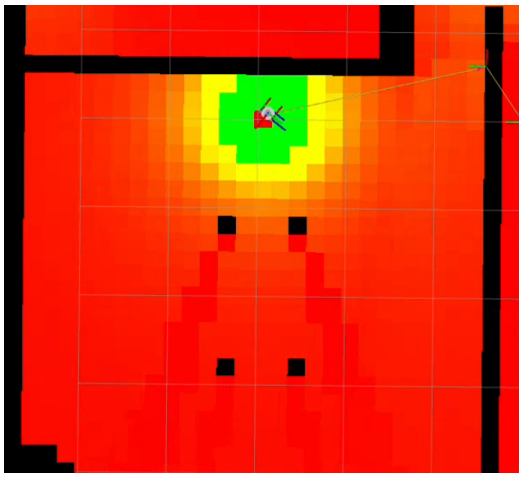
4.3.2 Energy Update

Suppose that the robot has a lamp able to spread all around it a light with a certain power P_{lamp}, the the UV energy E at a certain point (x, y) and time t can be computed, in discrete time, as:

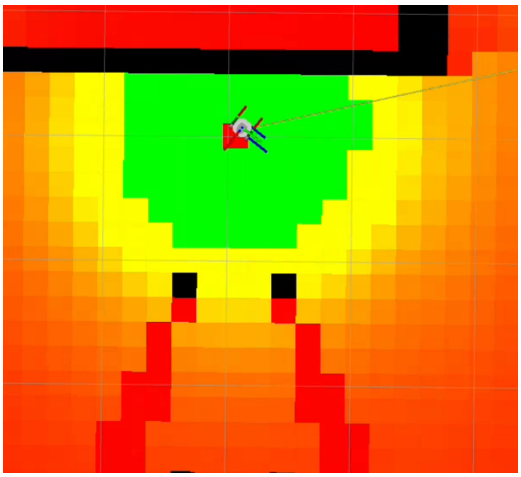
$$E(x, y, k) = \sum_{t=0}^{k} \frac{P_{lamp} \cdot \Delta t}{(x - x_{rob})^2 + (y - y_{rob})^2} \tag{2}$$

Obviously the energy can be updated only if the cell is visible. So basically we have created a timer which call every Δt seconds a function that increases the energy of a cell considering whenever the following constraints are respected:

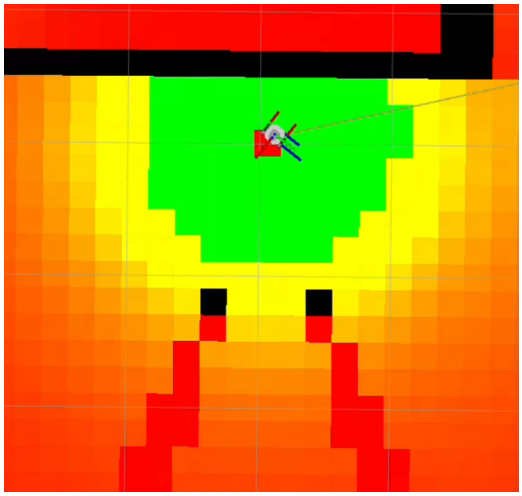
- Lamp = ON
- Cell Distance > 10cm. That means that the cell is not covered by the robot since it's radious is 10 cm.
- Cell Visibile = True. Only the visible cells can increase the energy



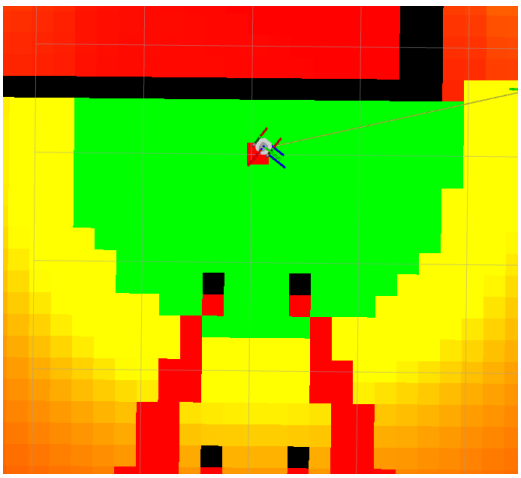
(a) Energy Spread



(b) Energy Spread



(c) Energy Spread



(d) Energy Spread

4.4 Planner Node

The planner node aims to generate a suitable path for the Sanitization task. In this node al the information about the room location are stored. For each room two opposite points of the rectangle are collected. Here it is possible to choose which room must be sanitised and the planner will generate the sequence of goals necessary to accomplish the task.

First of all whatever is the position of the robot, the planner set a goal to the center of the room to be sanitised and generate an "S-Shaped path" to cover all the room. The "S-Shaped" path is written in a text file when the center of the room is reached. After that a topic called "/navigate" is set to true by this node. This topic will be used by the Navigation node in order to understand when it should start to navigate for the Sanitization.

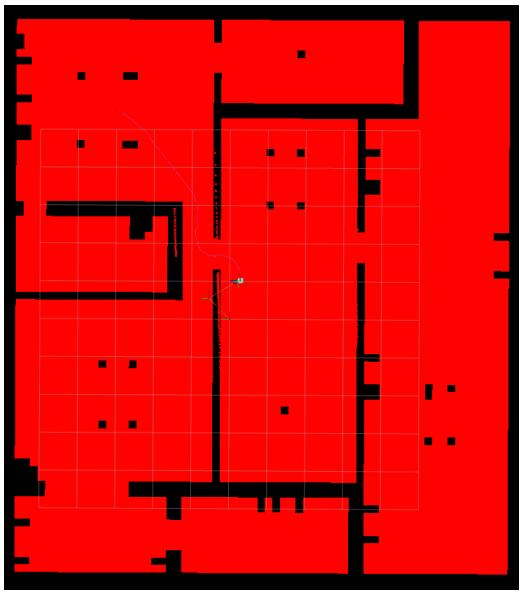
Figure 13: Goal Planner

In figure 13 it's possible too see how the algorithm generates the goals. It search for the longest side of the room and set it as the "main" direction of navigation. Once the end of the room is reached, set the goal in the "next row" and then reach the other side of the room. The navigation node it's responsible to avoid goals to be pusblished over obstacles.

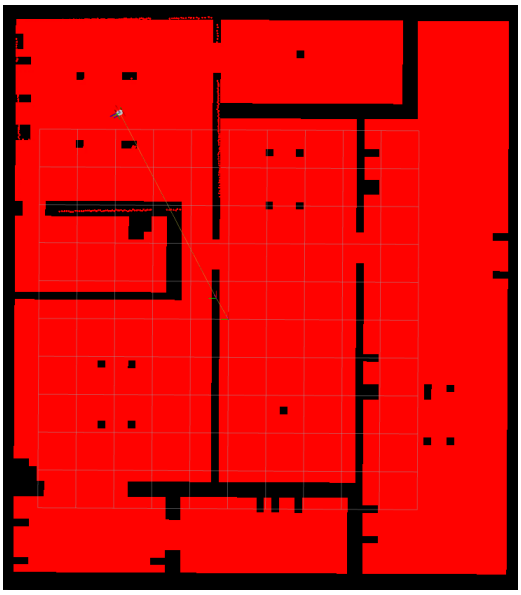
4.5 Navigation Node

The Navigation Node discussed in section 3 has been adapted in order to "understand" when it's time to navigate and consequently when it's time to turn ON or OFF the lamp. The node is still particularly simple: whenever the planner node set the topic "/navigate" to true, this happen when the planned path has been written on a text file, this node read the goals and set the topic "/lamp" to true. Then the goals are reached once at time. Since the lamp now its on the energy from the sanitizer node is spread all over the visible cells. When the last goal of the planner is reached the lamp is turned OFF. By combining the functions of the three nodes, it is possible to achieve the desired room sanitization.

In the following figures it's shown how, starting from the room denoted as "Room 5" the robot goes to "Room 3" and sanitise that room.

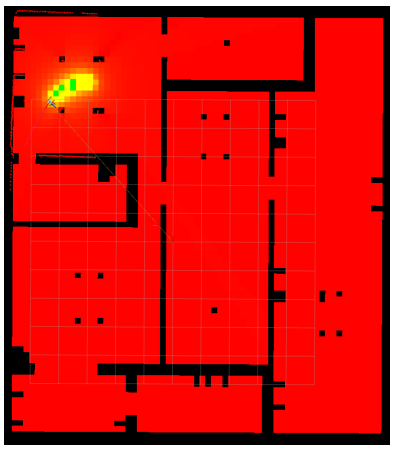


(a) Robot start from room 5

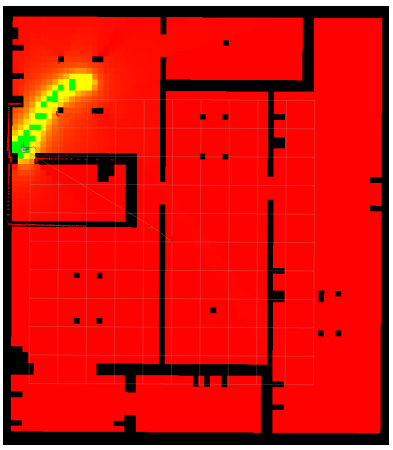


(b) Robot reach room 3

While the robot reach the desired room, the lamp is turned OFF, the energy is not spread. The following images shows what happens when the center of the room is reached.

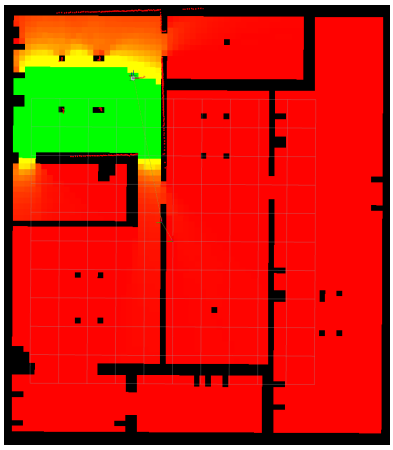


(a) Lamp Turn ON



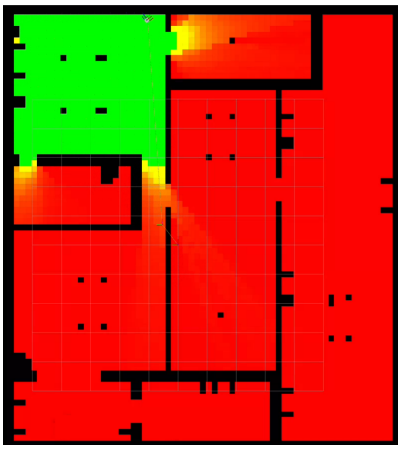
(b) Reach first goal

(c) Other goals...



(d) Other goals...

(e) Other goals...



(f) Sanification Completed

Up to now we can set just one room at time to sanitise from the planner, however a further implementation could be to set a set of rooms and to sanitise them sequentially.

Remarks

4.6 Extended Common Visibility Points Technique

The common visibility points technique could be extended: instead of linking the robot and the candidate goal through a single common point, that could be done with an arbitrary number of points, this could be useful to find goals behind multiple layers of obstacles that are not possible to find with a single common visibility point. Picking the situation shown in Figure 7, the problem could be solved finding these 2 common visibility points instead of exploiting Backtracking:

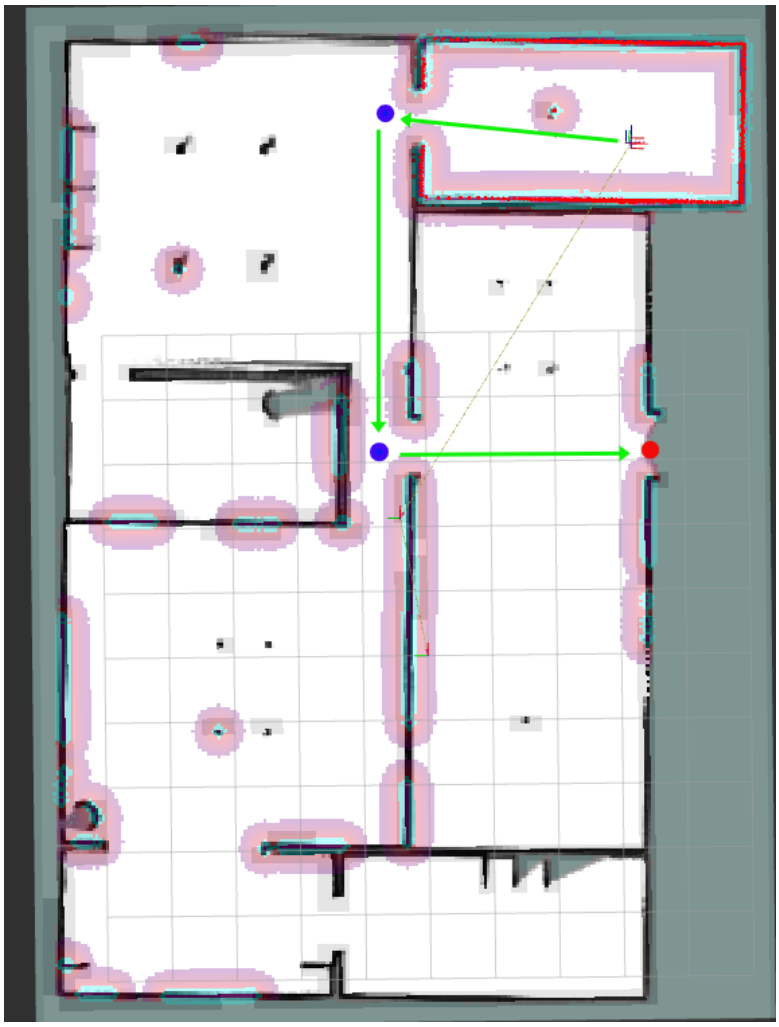


Figure 16: Visualization of multiple common points (blue) and goal point (red)

Although this extended common visibility point technique could be very effective, it is probably more computationally complex than backtracking.