Development of Robotic Agents for Environment Mapping and Cleaning

João Clemente[1], jclemente@ua.pt

DETI, Universidade de Aveiro, Portugal

Abstract. This report outlines the development of two robotic agents capable of mapping and cleaning an environment using a simulated MiR 100 robot platform enhanced with a virtual vacuum cleaner bar. The first agent goal is to build a complete map of the environment in the minimum time possible, while the second agent should clean the generated environment efficiently.

Keywords: Robotic Agent · SLAM · ROS.

1 Introduction

This project aims to develop two autonomous robotic agents that perform tasks of environmental mapping and subsequent cleaning using the Robot Operating System **[1]**(ROS). The agents leverage the Mobile Intelligent Robots (MiR) platform, demonstrating advanced capabilities in autonomous navigation and operational efficiency.

2 Implementation

The development of the project was done using available ROS packages that required the proper changes in order to make them efficiently to tackle the defined goals. Some of them were provided by the teacher in the lectures' lessons and the remaining were found on-line.

In this section, the core part of the working implementation that was done will be explained.

2.1 Mapping Strategy

After downloading all the proper documentation, the main initialization nodes and packages were gathered in the same launch file, in order to make it easier to start the simulation.

As shown below, the *1_part_exploration.launch* file firstly initializes the gazebo world alongside with the pre-defined map and the robot.

The *move_base.launch* allows the robot to steer.

Listing 1.1: ROS launch file for map exploration.

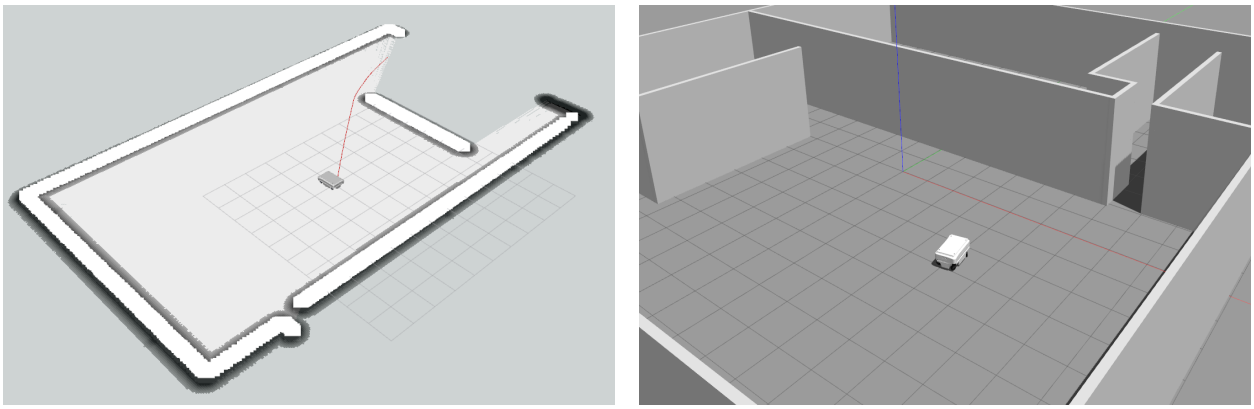
```
<!-- Exploration part of the project -->
<launch>
  <!-- Launch the simulation with the steering robot app and simple
      environment -->
  <node name="start_world" pkg="rm_mir_cleaner" type="start_world.sh"
      output="screen">
    <param name="use_sim_time" value="true"/>
  </node>

  <!-- Launch the map server and pre-existing map -->
  <include file="$(find␣rm_mir_cleaner)/launch/start_map.launch"/>

  <!-- Launch the move base framework -->
  <include file="$(find␣rm_mir_cleaner)/launch/move_base.launch"/>
```

```
<!-- Launch the merging scans file -->
<include file="$(find␣rm_mir_cleaner)/launch/laserscan_multi_merger.
    launch"/>

 <!-- Launch the SLAM node (gmapping example) -->
  <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping"
      output="screen">
    <remap from="scan" to="/scan_multi"/>
    <param name="r" value="3.0"/>
    <param name="linearUpdate" value="0.5"/>
  </node>

<!-- Start RViz with a specific configuration file -->
<node name="rviz" pkg="rviz" type="rviz" args="-d␣/home/lclem0/RM/
    rm_cleaner_ws/1_part_exploration.rviz" required="true"/>

<!-- Launch explore_lite for autonomous exploration -->
<include file="$(find␣explore_lite)/launch/explore.launch"/>

</launch>
```

To implement the proper mapping of the environment, several addings were done. Firstly, after loading the Rviz environment with the proper visualization tools, to better understand what was happening, we had to merge the front and back scanner that the MiR robot has.

In order to do this, the *ira_laser_tools* package from **[2]** was used. The *laserscan_multi_merger.launch* file was adapted and integrated into the *rm_mir_cleaner* package.

As seen below, the *f_scan* and *b_scan* were merged into *scan_multi* topic, that enabled a more comprehensive understanding of the complete environment around the robot, as depicted in the figure 1a, as the white flat squares on top of the walls.



(a) Rviz view from merged laser scans

(b) Gazebo view of the environment

Fig. 1: Difference between the enabled laser merger visualization in Rviz and Gazebo

Listing 1.2: Laser scan multi merger

```
<node pkg="ira_laser_tools" name="laserscan_multi_merger" type="
    laserscan_multi_merger" output="screen">
        <param name="destination_frame" value="base_link"/>
```

```
        <param name="cloud_destination_topic" value="/merged_cloud"/>
        <param name="scan_destination_topic" value="/scan_multi"/>
        <param name="laserscan_topics" value ="/f_scan␣/b_scan" /> <!
            -- LIST OF THE LASER SCAN TOPICS TO SUBSCRIBE -->
        <param name="angle_min" value="-3.14"/>
        <param name="angle_max" value="3.14"/>
        <param name="angle_increment" value="0.0058"/>

                <param name="scan_time" value="0.0333333"/>
        <param name="range_min" value="0.30"/>
        <param name="range_max" value="50.0"/>
</node>
```

In order to perform the simultaneous localization and mapping (SLAM) of the environment, several recommended tools were given in the lectures, such as *gmapping*, *hector_mapping*, *iris_lama_ros* and *cartographer*.

To developed the project, it was used the gmapping **[4]** from the previously defined topic */scan_multi*.

```
  <!-- Launch the SLAM node (gmapping example) -->
<node pkg="gmapping" type="slam_gmapping" name="slam_gmapping"
    output="screen">
  <remap from="scan" to="/scan_multi"/>
  <param name="r" value="3.0"/>
  <param name="linearUpdate" value="0.5"/>
</node>
```

The Rviz was used alongside with Gazebo, to more clearly see the behaviour of the robot when traversing the map. It was created once and then saved in order to load it everytime the simulation was done. The used file was the *1_part_exploration.rviz* and its appearance it visible in the figure 2. It has the following displays:

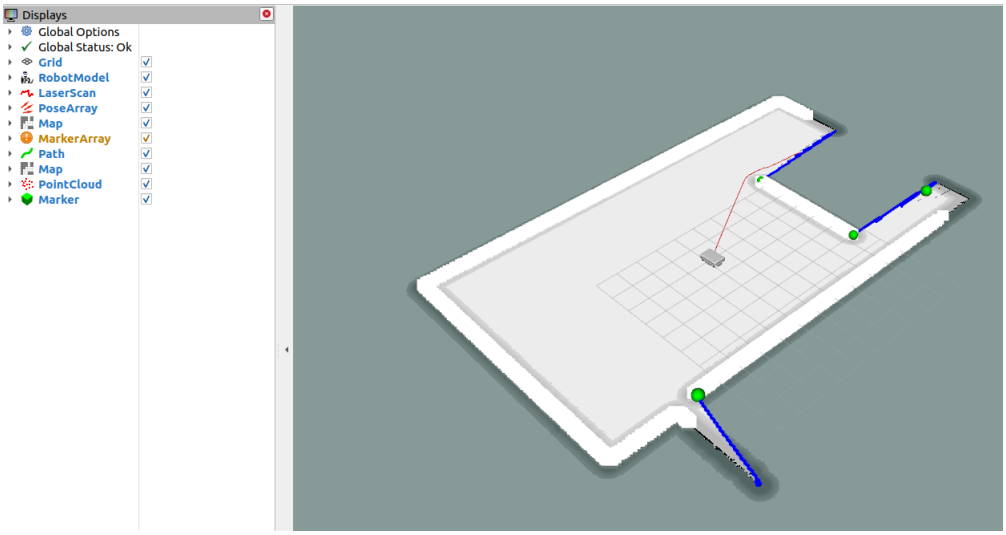
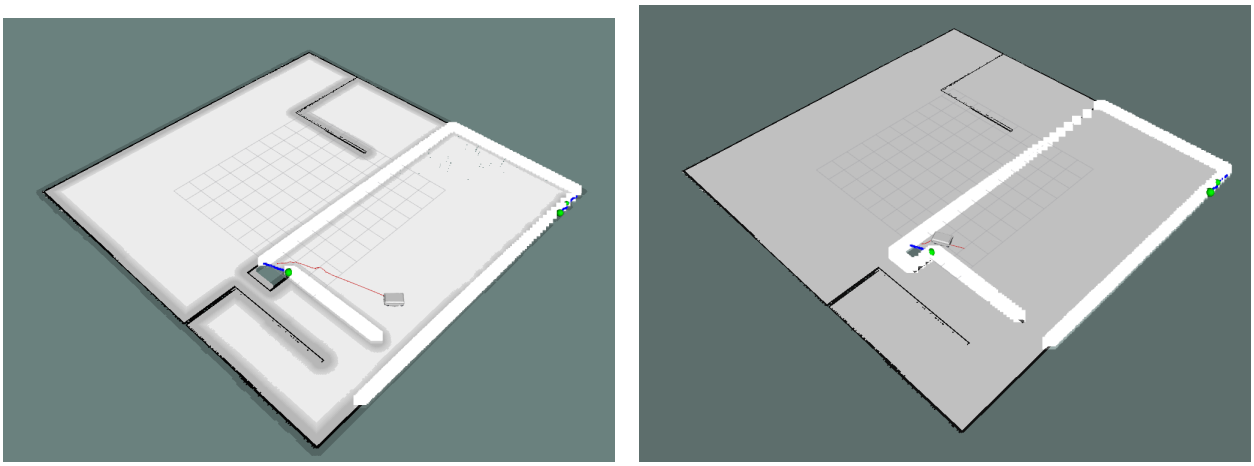


Fig. 2: Rviz displays used in the loaded *1_part_exploration.rviz* environment

– **Map Visualization:** Both the */map* and the */move_base/global_costmap/costmap* are visible. The global costmap provides insight into the traversal cost of different regions within the environment, high-

lighting areas near the walls as gray clouds. Differences between the maps are illustrated in subfigures 3a (with costmap) and 3b (without costmap), shown in figure 3.



(a) Visualization of map with global costmap in Rviz (b) Visualization of map without global costmap in Rviz

Fig. 3: Differences between Rviz visualization using the global costmap 3a and not using the global costmap 3b

– **Marker Array:** The $explore/frontiers$ topic generates an array of markers, which helps in identifying unexplored areas. These unexplored regions are indicated by green spheres and blue lines, as depicted in figure 4.

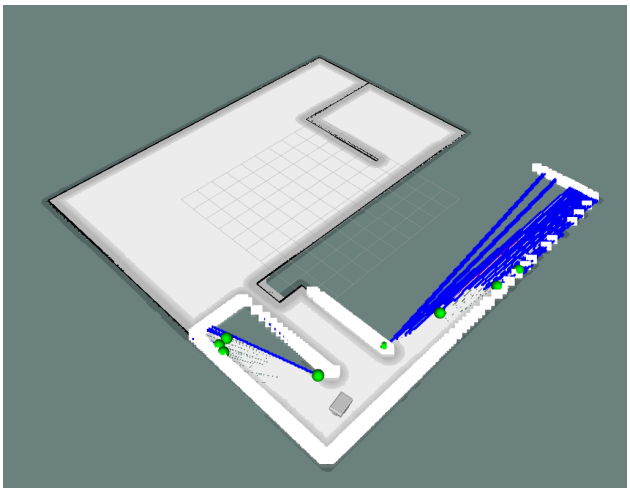


Fig. 4: Rviz Marker Array of unexplored areas of the map

– **Path:** The $/move\_base/DWBLocalPlanner/global\_plan$ topic generates the desired path the robot will make based on its planner. In this case it is used the DWB Planner, that will further below explained. In the figure 5 it is possible to see it represented as a red line.

Regarding the $move\_base.launch$, which is responsible for combining both the global planner and the local planner, it moves the robot around obstacles in real-time, also visible in Rviz.

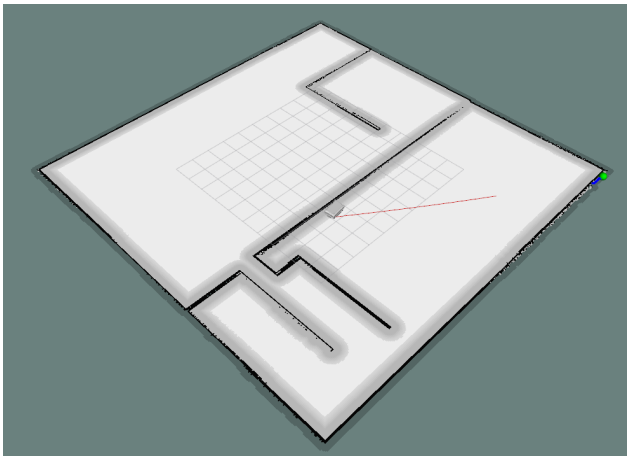


Fig. 5: Path Visualization in Rviz

Listing 1.3: Launch File for Move Base Node

```
<launch>
    <arg name="local_planner" default="dwb" doc="Local␣planner␣can␣be␣
        either␣dwa,␣base,␣teb␣or␣pose"/>
    <arg name="prefix" default="" doc="Prefix␣used␣for␣robot␣tf␣frames
        " /> <!-- used in the config files -->

    <node pkg="move_base" type="move_base" respawn="false" name="
        move_base" output="screen"          clear_params="true">
        <param name="SBPLLatticePlanner/primitive_filename" value="$(
            find␣mir_navigation)/mprim/unicycle_highcost_5cm.mprim" />

        <rosparam file="$(find␣mir_navigation)/config/
            move_base_common_params.yaml" command="load" />

        <!--  costmap params -->
        <rosparam file="$(find␣mir_navigation)/config/
            sbpl_global_params.yaml" command="load" />
        <rosparam file="$(find␣mir_navigation)/config/$(arg␣
            local_planner)_local_planner_params.yaml" command="load" /
            >

        <!-- global costmap params -->
        <rosparam file="$(find␣mir_navigation)/config/
            costmap_common_params.yaml" command="load" ns="
            global_costmap" subst_value="true" />
        <rosparam file="$(find␣mir_navigation)/config/
            costmap_global_params_plugins_no_virtual_walls.yaml"
            command="load" />

        <!-- local costmap params -->
        <rosparam file="$(find␣mir_navigation)/config/
            costmap_common_params.yaml" command="load" ns="
            local_costmap" subst_value="true" />
        <rosparam file="$(find␣mir_navigation)/config/
            costmap_local_params.yaml" command="load" subst_value="
            true" />
```

```
        <rosparam file="$(find␣mir_navigation)/config/
            costmap_local_params_plugins_no_virtual_walls.yaml"
            command="load" />
        <remap from="map" to="/map" />
        <remap from="odom" to="odom_comb" />
        <remap from="marker" to="move_base_node/DWBLocalPlanner/
            markers" />

    </node>
</launch>
```

It allows for the specification of different local planners through the local_planner argument, such as *dwa*, *dwb*, *base*, *teb* or *pose*.

In this case it was used the *dwb* which proved to be more efficient related to the remaining ones, which revealed unwanted behaviors such as too much jittering when performing the planning.

According to the costmaps configuration, despite trying to change some of its values, both on local and global costmap parameters are the base ones from the *mir_robot* github **[2]**.

As for the autonomous exploration of the map, the *explore_lite* package **[5]** was used and the defined properties can be seen below.

Listing 1.4: ROS Launch Configuration for *explore_lite*

```
<launch>
    <node pkg="explore_lite" type="explore" name="explore" output="
        screen">
        <param name="robot_base_frame" value="base_link"/>
        <param name="planner_frequency" value="0.2"/>
        <param name="potential_scale" value="0.1"/>
        <param name="orientation_scale" value="0.2"/>
        <param name="gain_scale" value="1.0"/>
        <param name="track_unknown_space" value="true"/>
        <param name="visualize" value="true"/>
        <param name="progress_timeout" value="30.0"/>
        <param name="transform_tolerance" value="0.3"/>
        <param name="min_frontier_size" value="0.5"/>
    </node>
</launch>
```

Both the *planner_frequency* and the *potential_scale* were reduced from 1.0 to 0.2 and 0.2 to 0.1, respectively. This *potential_scale* change has effects on the potential field in the costmap used for planning. The *orientation_scale* remained the same and some other parameters were added.

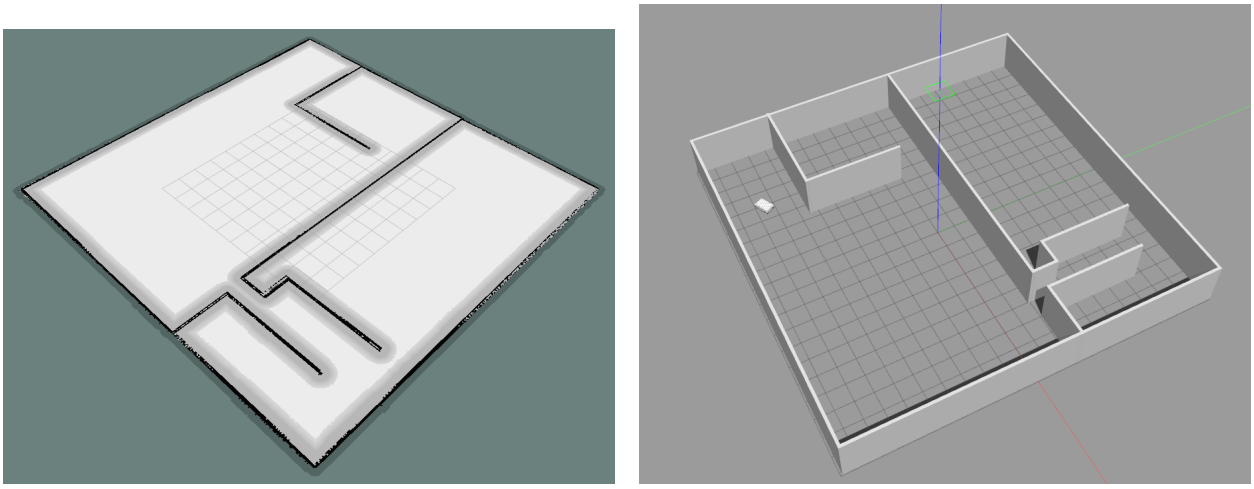
Given this optimized launch file, the first part of the project is simulated using only

```
roslaunch rm_mir_cleaner 1_part_exploration.launch
```

which traduces in a good mapping performance for the gazebo map of the figure 6b, represented in the figure 6a

Listing 1.5: ROS Launch Configuration for maze gazebo model

```
<include file="$(find␣mir_gazebo)/launch/mir_maze_world.launch">
  <arg name="gui" value="$(arg␣gui)" />
  <arg name="tf_prefix" value="$(arg␣tf_prefix)" />
</include>
```



(a) Final map from concluding first simulation part (b) Gazebo maze model used in the simulation

Fig. 6: Generated map from first part simulation 6a and its corresponding Gazebo model 6b

2.2 Cleaning Strategy

Now, into the second part of the project which consists on cleaning the environment, the robot most traverse the majority of the free space and clean it.

The following bash command runs it

```
roslaunch rm_mir_cleaner 2_part_cleaning.launch
```

.

Despite being able to save the first part generated map into an .yaml file, it was not possible to launch it efficiently into Rviz.

Several maps were saved after running completely the first part by using the command

```
rosrun map_server map_saver -f "desired_map_name"
```

and saved, therefore, into the project *maps* folder.

Given the issue of not loading them correctly into the simulation, this part's initialization was done similarly to the previous one.

The *move_base.launch* file was modified into the *move_base_part_2.launch*, whose change consisted on not using the

```
<rosparam file="$(find mir_navigation)/config/sbpl_global_params.yaml" command="load"/>
```

because it generated useless circular trajectories, as shown in the figure 7.

Afterwards, the part responsible for cleaning the environment is started.

Listing 1.6: ROS Launch Configuration to clean the environment

```
<node name="rm_cleaner" pkg="rm_mir_cleaner" type="rm_cleaner"
    output="screen">
</node>

<include file="$(find␣path_coverage)/launch/path_coverage.launch"/>
```

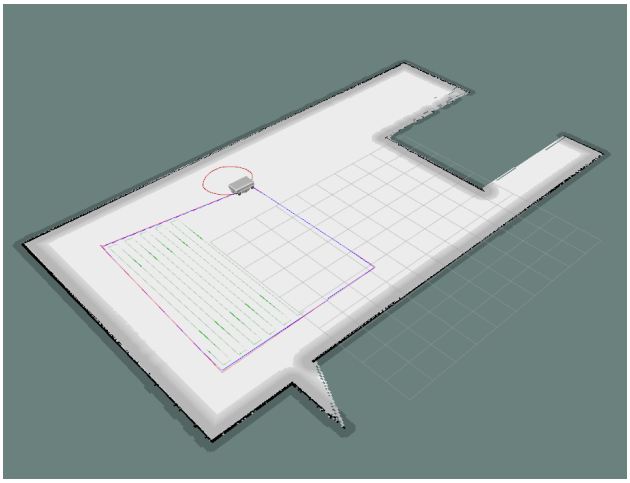


Fig. 7: Circular Trajectories with *move_base* from first part of the project

The *rm_cleaner* node launches the base code that marks the position of the robot in the loaded map, visible in the *rm_cleaner.png*, at the end of the second part of the simulation.

The *path_coverage* package, was imported from Gitlab **[6]** and allows for the specification of the area the robot must traverse in order to clean it.

However, since it was not possible to instantly load the generated map into the rviz, the cleaning process of the robot has to be done mannualy.

This is done by publishing points in the Rviz interface until generating a closed polygon. Then, the robot proceeds to traverse this closed polygon according to the *boustrophedon_decomposition*.

Two different generated polygons are shown in the figure 8, where the top subfigures show the planned path of the robot for a bigger area of the map. It cleans it linearly alonside the biggest path and then rotates a bit to the side and repeats it again until ending all of the cleaning process. Below, the bottom figures show the path the robot plans to the starting point of the cleaning area, if it is not there yet. Only after reaching it, it starts the cleaning process.

3 Results

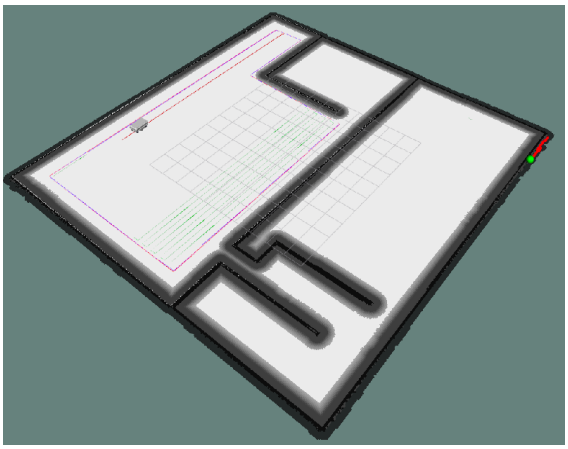
Both agents for the first and second part were implemented. Regarding the results obtained, the developed strategy has been already explained with the corresponding insights.

The first part's goal was achieved more efficiently since the mapping of the environment was done for different parameter tuning of the *move_base* and *explore* launch files. Despite obtaining different running times, it always ended with a full map result. There was a issue when developing this first part, because it does not work right away the first time it is run. If stopped and relaunched, it will get no error while running. The origin of this problem was not found, but it is suspected to have to deal with the order that the packages are launched in the main *1_part_exploration.launch*.

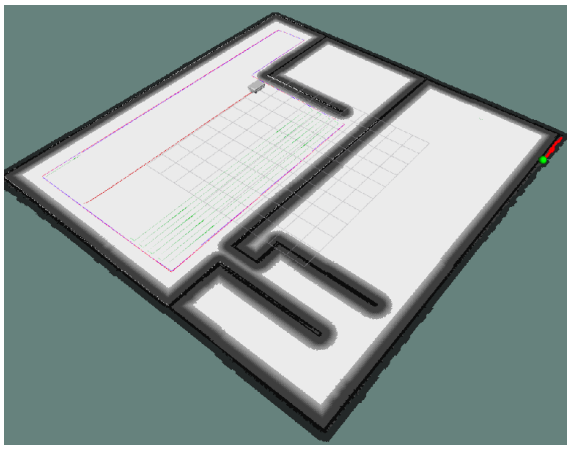
On the other hand, the development of the second part of the project was more difficult since the loading of the saved map from the prior part lead to some issues. During its running time, the planning of the path to traverse, sometimes returns an error. The subscribed topic is the */odom* instead of the supposed one, */map*. Despite trying to solve this error, it was not possible to do so. Therefore, the final image after cleaning the free space, does not show the area that the robot cleaned. It is believed that both these issues would be solved if the loading of the generated map was being done properly.

4 Conclusion

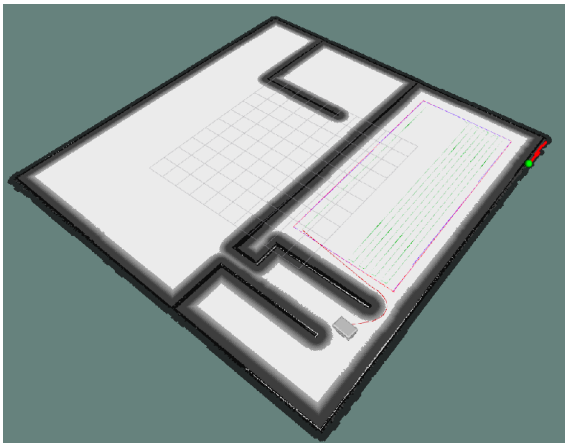
In conclusion, the development of the project allowed to gain a comprehensive understanding of how to implement robotic agents for environment mapping and cleaning, using ROS.



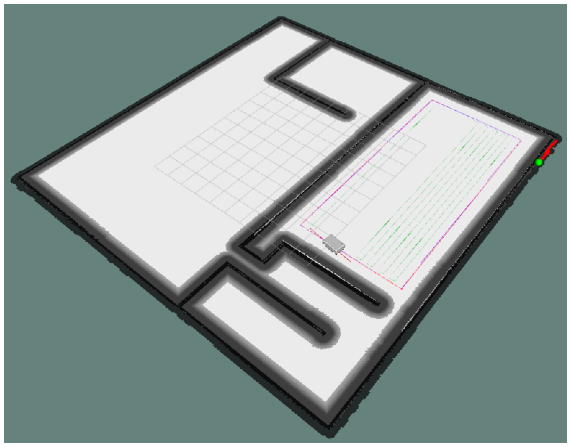
(a) Area Traversing polygon 1



(b) Area Traversing polygon 1 - path 2



(c) Area Traversing polygon 2



(d) Area Traversing polygon 2 - part 2

Fig. 8: Path comparison for different traversing polygons in the maze map.

The first part of the project, that consisted in mapping all of the environment, specially a more difficult map than the first proposed one, was done successfully. Different parameter tuning was done and the most efficient is shown.

For the second part of the project, some issues arose but they represent a slightly obstacle into the full completion of the autonomous cleaning agent. Given that the loading of the map could not be achieved, this resulted in having to mannually specify the areas that the robot must traverse, as well as not being able to fully visualize which parts of the map it cleaned in the *rm_cleaner.png* file.

Therefore, it is possible to conclude that the main goals of the project were achieved.

References

1. ROS.org. *Robot Operating System*, http://www.ros.org
2. DFKI-NI. *mir_robot*, available at https://github.com/DFKI-NI/mir_robot.
3. IRA Laser Tools, http://wiki.ros.org/ira_laser_tools, also available as an Ubuntu package.
4. Gmapping, http://wiki.ros.org/gmapping, also available as an Ubuntu package.
5. explore_lite, http://wiki.ros.org/explore_lite, also available as an Ubuntu package.
6. Humpelstilzchen. *path_coverage_ros*, available at https://gitlab.com/Humpelstilzchen/pat_coverage_ros.