Report - Underwater Systems Project

Group 2A

Anthony Cappetta, Giacomo Galli, Emilio Gigante,
Bianca Malucchi

Indice

1 Introduction

This project aims to address an obstacle avoidance problem. Specifically, the project consists of enabling an underwater vehicle, equipped with a multi-beam sonar, to navigate autonomously within an environment characterized by the presence of obstacles, and to reach given target positions by planning its path and avoiding collisions. Consequently, the project is divided into three parts.

1. The first part addresses the use of measurements provided by the sonar to create and update in real time a map of the surrounding environment and identify any obstacles within it.
2. The second one models the sonar functions, which must provide measurements of the distance between the vehicle and detected obstacles.
3. The third and last part implements a two-dimensional path planning strategy based on the reconstructed map, in order to reach the selected goal position without hitting obstacles.

2 Structure of the work

The implementation of our work was made using the platform provided by Robot Operating System (*ROS*). All the groups wrote one or more *ROS* nodes containing the algorithms necessary for their respective tasks. These nodes then communicate with each other and the vehicle (Zeno) using appropriate *ROS* messages published on specific *ROS* topics. An overview of this structure organization is shown below.

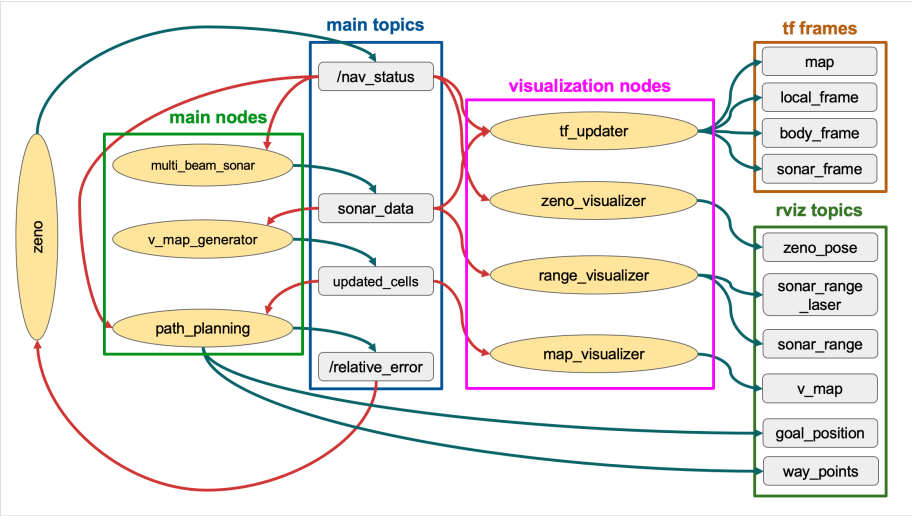

Figura 1: Scheme of the Flow of the Data

Each subgroup worked on one of three main nodes: (1) to model the behaviour of the sonar, (2) to create and update the virtual map, and (3) to implement the path planning algorithm.

These main nodes interact with Zeno using the topics `\nav_status`, which contains the information about the current position and attitude of Zeno, and `\relative_error`, where the information about the control, in terms of yaw angle and surge speed, is published.

In addition, there are some secondary nodes that take care of visualisation. These nodes update the rotation matrices between the various frames used and publish information about Zeno, the sonar and the virtual map on specific topics. This information is then used by *Rviz* to display the progress of the ongoing mission in real time.

2.1 Parameters

Some "global" parameters are used by more than one group at the same time. For this reason, these parameters have been defined in a specific script, outside the main nodes, so that all nodes can simply import this file and access the required information. In particular, these parameters are:

- The number of the sound beams provided by the multi-beam sonar.
  `Experiment value: 15`
- The range of the sonar, which represents the maximum distance from the device within which obstacles are detected (in meters).
  `Experiment value: 10 m`
- The angular range in which all the beams are placed (in radians). Since all the beams are assumed to be equally spaced, this allows us to calculate the angle between two adjacent beams.
  `Experiment value: 120°`
- The resolution of the grid representing the virtual map (in meters).
  `Experiment value: 0.5 m`
- The radius of Earth (approximated to a sphere), in meters, in the area where the experiment was conducted.
  `Experiment value: 6368000 m`
- The margin (in meters) around the cell in which an obstacle is detected. All cells within this margin are assigned a certain uncertainty value regarding the presence of an obstacle.
  `Experiment value: 1 m`
- The resolution of a cell in decimal degrees of latitude and longitude. This is used to translate the position of the vehicle, which is given to us in terms of longitude and latitude, to our local frame.
  This is calculated as
  $$deg\_resolution = \frac{180 * resolution}{\pi * (Earth\_radius)}$$

3 Our Node

In this section, we will focus on explaining how the node works. In particular, we (group 2A) will see what kind of message and data we receive from group 2B, the algorithm used to update the virtual map, and the data sent as output to group 2C.

3.1 Received data

As mentioned earlier, Group 2B is tasked with modeling the behaviour of a multi-beam sonar. Specifically, they read a .txt file containing information regarding the actual map and the position of obstacles. More specifically, the information is reported following the syntax of the *Python* library *Shapely*. Regarding the real map, the file contains the positions (in terms of latitude and longitude) of the map vertices, such that the resulting map is nothing but the polygon connecting those vertices. Regarding the obstacles, if they are polygonal, information about the position of the vertices is provided (longitude and latitude), while for circular obstacles are given the position of the center and the radius in meters.

By knowing the information contained in the .txt file and knowing the position and orientation of Zeno subscribing to the topic `\nav_status`, they are able to associate with each of the sonar beams, that will be headed in different directions, a value indicating the distance at which that beam encountered an obstacle[1], or, if it encountered none, a fixed value, which in our case was set to -1.

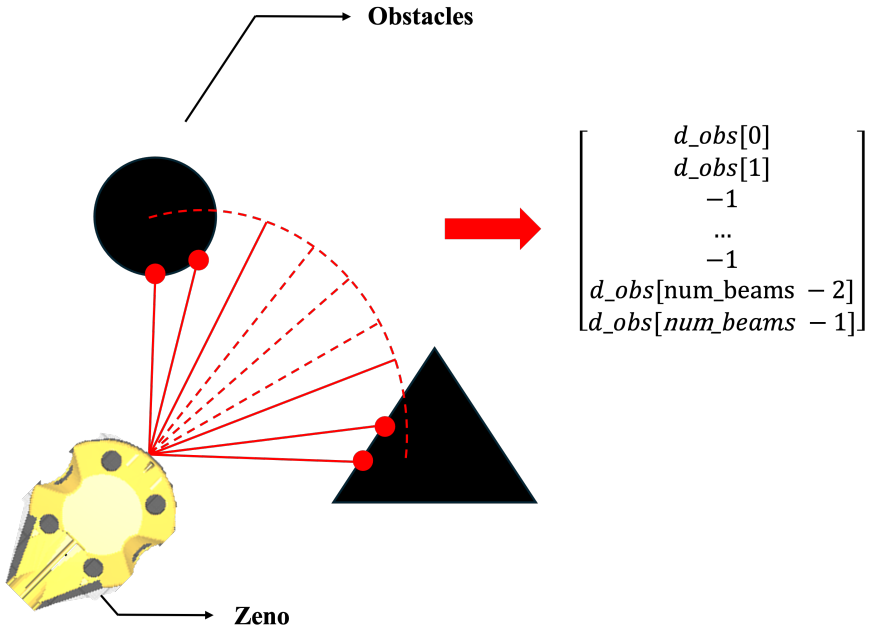

Figura 2: Array of the distances provided by the sonar

[1]Note that the borders of the real map are also considered as obstacles

This information is provided to us via a custom ROS message, `MySensor.msg`, posted on the topic `\sonar_data`, to which our node subscribes. This message includes an array, equal in size to the number of beams, that contains information regarding the distances associated with each beam. In particular, the first element refers to the leftmost beam, and from then on clockwise.

Moreover, the message also contains the longitude, latitude and yaw angle of Zeno at the time the measurement was taken. In this way, the position of the obstacles (by means of an appropriate coordinate change function) can be derived with reference to our local frame, which is a NED frame with origin in the vertex with the lowest longitude and latitude of our virtual map.

3.2 Our algorithm

The crucial part of our work is the implementation of the algorithm responsible for updating the virtual map in real time, based on the measurements obtained from the sonar.

3.2.1 Initializing

When the node is launched, we need to read the .txt file to establish the boundaries of the real map and create the virtual map, accordingly. By reading the file, we can derive the longitude and latitude of the map vertices, and between them find the minimum and maximum longitude and latitude. Consequently, the virtual map is created as the rectangle that inscribes the real map, as shown in Fig. 3.

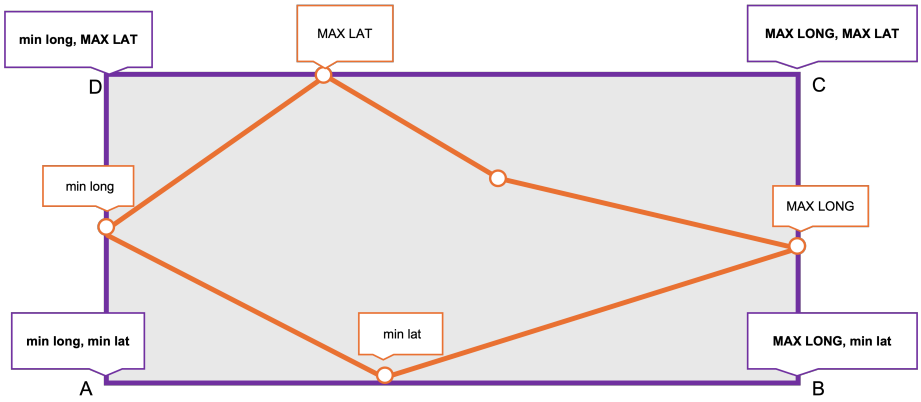

Figura 3: Example of the borders of the real map (orange) and the ones of the virtual map (purple)

By means of the difference between the maximum and minimum longitudes and latitudes, and only after a change of coordinates to return to metres, the width and height of the virtual map can be calculated. Additionally, using the resolution of the map (which in our case is set as $0.5m$) makes it possible to derive the number of rows and columns of the matrix representing this virtual map.
In fact, the reconstructed map has been implemented as a matrix of integers. In this matrix, each cell represents a square with a side equal to the chosen resolution. The value of the individual cells represents a kind of probability of having an obstacle within it.
Specifically, the value 0 corresponds to a free cell, where no obstacle is present. Contrasting, the maximum value that each cell can assume, and which represents the maximum certainty of the

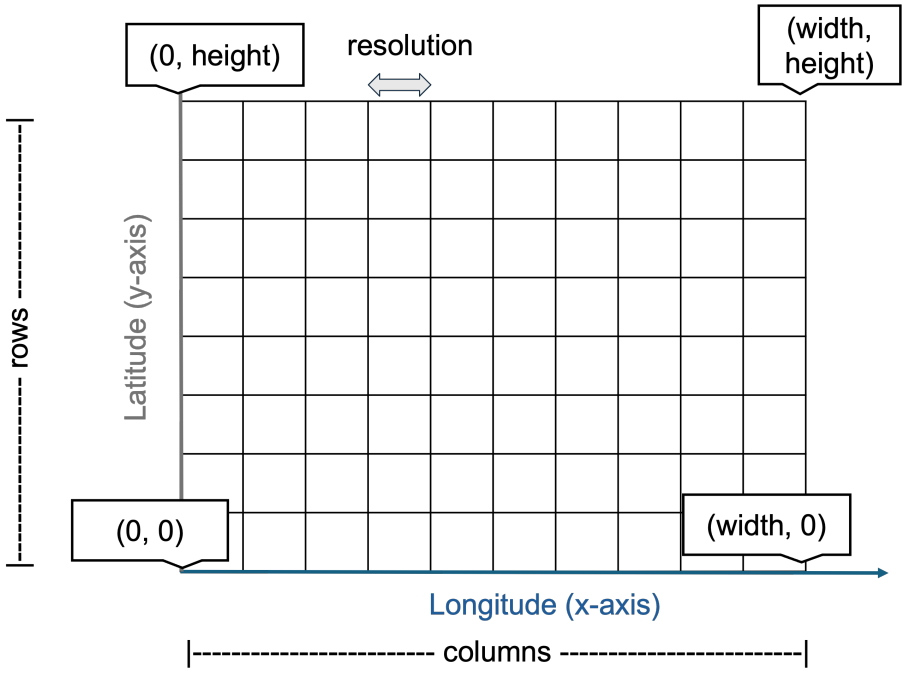

Figura 4: Example of the matrix representing the virtual map

presence of an obstacle in that cell, was taken on the basis of the number of beams (in particular, for our experiment this value was set as $5 * num\_beams$).

The matrix is initialized to zero, which implies that initially the map is considered to be all clear and the vehicle is free to navigate anywhere, at least until further updates detect the presence of obstacles.

3.2.2 Updating

The algorithm that updates the virtual map based on sonar measurements is contained in the callback function that is called each time a new message is posted on the topic `\sonar_data`. In this way, the update is performed each time a new measurement is available, using the data related to it.

Sequentially, the information about Zeno's longitude, latitude and yaw angle at the time the measurement was taken, contained in the incoming message, is then saved. Zeno's position is then converted into Cartesian coordinates with respect to the local frame. Having saved the vector of distances `d_obs` contained in the message and knowing from the parameters the angle between two consecutive beams, the matrix update can begin. This is completely cyclic.

For each beam, we know its orientation with respect to the local frame and the distance associated with it. Therefore, we can distinguish two possible scenarios:

- If the beam encounters no obstacles, we interpret as free all the cells which that beam encounters in its path. To do this, the beam is divided into segments of fixed length (in our

case, the ratio of the sonar range to the map resolution) and check in which cells the ends of each segment are located. The value of these cells is then lowered, if it is not already zero, by a predefined amount `free_threshold` (which in our case was 1).

This is done so that, if an obstacle is mistakenly detected in a cell, after a certain number of times that the cell is detected as free, it can again be considered as truly free. The values of the `free_threshold` and the maximum possible cell value are taken to have a trade-off between the possibility of making cells free again without risking freeing cells in which obstacles are actually present[2].

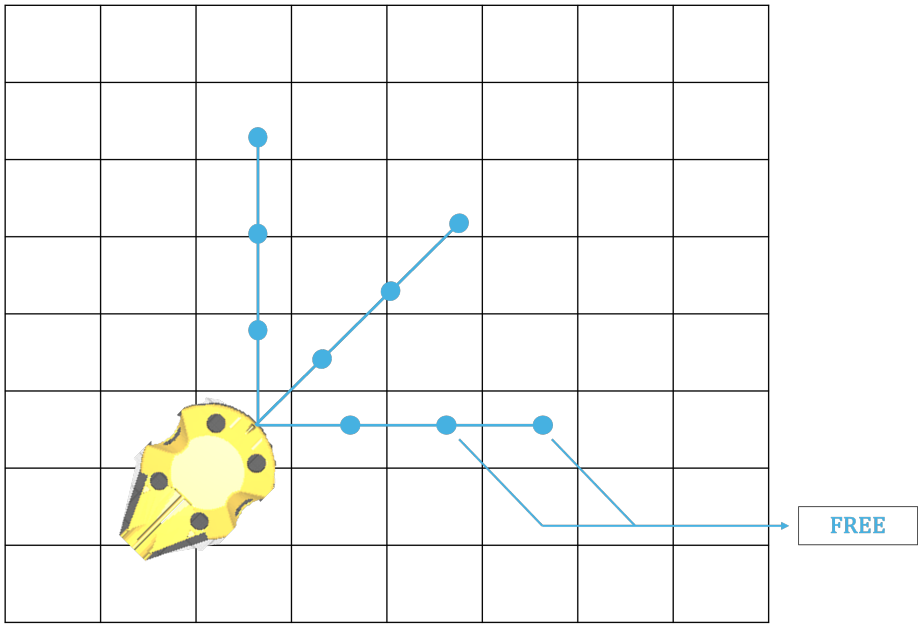

Figura 5: Scenario where the beams don't find any obstacles

- If the beam encounters an obstacle, the position of the point where it was detected in the local frame is calculated. Consequently, the cell in which this point lies is identified.

  The value of the cell in question is automatically set equal to the maximum possible value, corresponding to the maximum security of having an obstacle in that cell. In addition, the value of all cells within the previously defined safety margin is increased by an amount equal to half the maximum value (if the value of these cells is already greater than half the maximum value, it is also set equal to the maximum value).

  Noteworthy, every cell encountered by the beam, as long as it is outside the safety margin, is considered to be free. Its value is lowered similarly to how it was done for beams that did not encounter obstacles.

[2]Remembering that they must be integers to be consistent with the way the matrix is defined

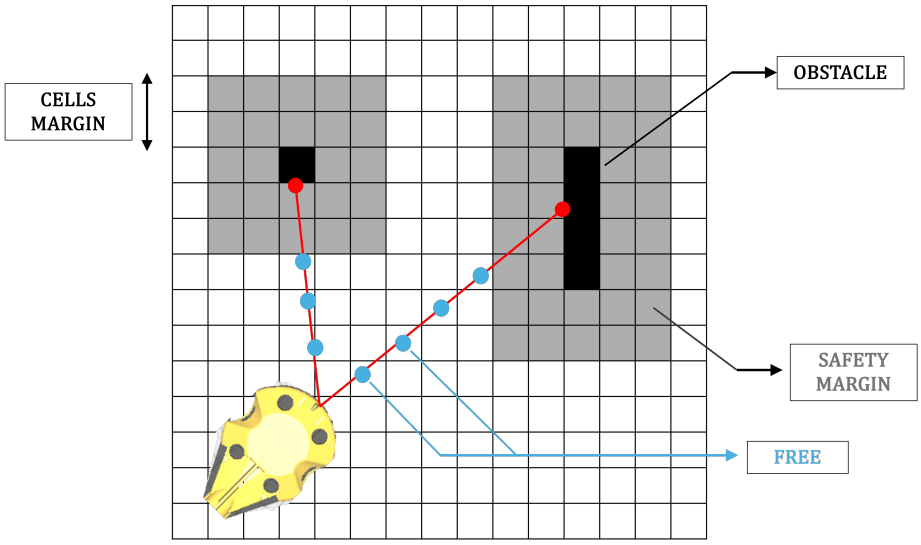

Figura 6: Scenario where the beams find obstacles

3.3 Shared data

At the end of each iteration, once all beams have been analyzed, the matrix was then updated according to the algorithm described above. To decrease the computational cost, however, we do not send the entire matrix information to group 2C each time, but we only send them the index of the cells which have changed their state from free to occupied and vice versa. For group 2C, in fact, it is not necessary to know precisely the probability of the presence of the obstacles in the cells. For security reasons, any cell that has a value other than zero is interpreted as occupied and is therefore considered an obstacle. For path planing, the third group will then use this data to create their own map, whose cells can only take the value 0 or 100 (free cell/obstacle).

To send them this information we used an additional custom message, `UpdatedCells.msg`, containing two arrays: one relating to the columns and the other to the rows of the cells to be updated. During the execution stage, whenever the value of a cell changes from 0 to a positive value, or vice versa, the row and column of that cell are saved in the relative arrays. As a result, at the end of each iteration, the message contains information about all the cells that have changed state. At the end of the callback, this message is published on the topic `\updated_cells`.

In addition, for visualization purposes, we also publish an `OccupancyGrid` type message (one of the standard *ROS* navigation messages). It contains information about the entire virtual map reconstructed up to that point. To prevent a large computational cost, this message is published on the topic `\uncertain_map` with a lower frequency specifically chosen (in this particular case, it was set to $0.1Hz$) and independently of when we receive the messages from group 2B. To do this, since the `OccupancyGrid` message requires cell values between 0 and 100, an additional map is created with values for each cell obtained by scaling those of the virtual map so that they fall within the desired range. This scaled map is then used to generate the `OccupancyGrid` to be published. This type of message allows us to display the reconstructed map, distinguishing actual obstacles and safety margin, using the *Rviz* software, with the cells being displayed on a grey scale according to

their value.

4 Final package

This section explains how the user can correctly manage the functioning of the *ROS* package we have developed.

4.1 Launch files

In order to start the mission, the main PC is connected to Zeno via WiFi, only needs to launch the main nodes (Fig. 1). A secondary PC, which launches visualization-related nodes, is also connected to Zeno. However, there is the possibility to launch all nodes simultaneously from the same PC. For this reason, it is useful to have three different launch files, one relating only to the main nodes, one to the visualisation nodes, and one comprising both groups of nodes.

Within the launch files, the initial goal position and other parameters are set, the purpose of which is explained below.

4.2 Commands

At the beginning of the mission, Zeno initially remains immobile. Through the use of parameters, two operating modes, 'STOP' and 'GO', have been provided. At the start, the mode is initialized to 'STOP' (in practice, 0 is given as the reference in the control topics). Via the command line, with the command `rosparam set`, it is possible to change the operating mode by modifying the parameter `\MODE_ZENO` and actually start the mission.

In addition, it is possible to change the goal position in the same way. This is meritorious to better explore the map. Once the current goal is reached, Zeno stops and automatically enters 'STOP' mode. Thereafter, it is possible to select a new goal and start a new mission.

All these parameters are stored in *ROS Parameter Server* to make it easier changing the settings during the mission.

4.3 Experiment recording

In order to record the messages posted on the various topics during the mission, and thus have data available from the experiment conducted, a *rosbag record node* is added to the launch file. This is a node already implemented in *ROS* that does exactly the job required for this purpose. In this way, we were able to save the information of interest within some .bag files. This allows one to use this information and data to redo simulations of the mission performed, obtaining the results that are shown in the following section .

5 Experimental Results

A practical test using the autonomous vehicle Zeno served as the project validation. During the test, we were provided with two different .txt files containing information about two maps, each containing various obstacles within it. These maps were provided following the syntax provided by the previously mentioned *Shapely* library.

The first map was presented as an 'easy' scenario, while the second one was intended to be a more demanding challenge in terms of quantity, shape and density of obstacles.

The maps were specially designed to be safely contained within the real lake. Given the following conditions, no boundaries or obstacles were physically present during the experiment, for safety reasons. The vehicle was controlled using the written nodes and was able to move within the lake, remaining just below the surface of the water so that it could be seen throughout the test.

The vehicle was initially deployed in the water and guided via a joystick to a predetermined starting point within the map. The *Rviz* software was used to monitor the progress of the mission in real time. The images shown below are obtained from the simulation of the experiment reproduced playing the *rosbag* recorded during the actual test.

5.1 Scenario 1: Easy Scenario

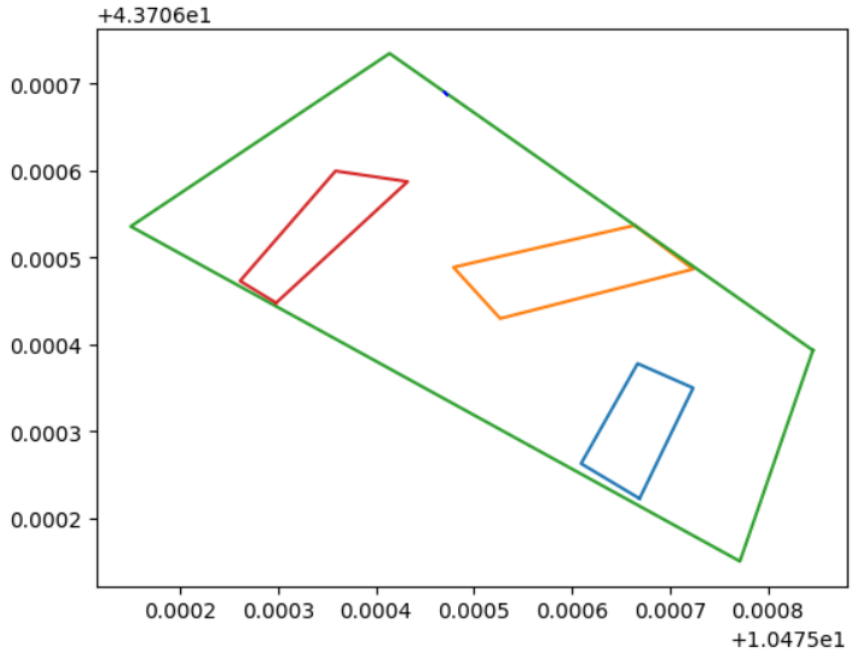

Figura 7: Representation of the map that was defined as an 'easy' scenario

Below are the coordinates of the map and the goal chosen for this scenario:

```
map = [(10.47584607, 43.70639344), (10.47541376, 43.70673506),
(10.47514949, 43.70653601), (10.47577151, 43.70615072),
(10.47584607, 43.70639344)]
#
polygon = [(10.47566904, 43.70622286), (10.47560934, 43.70626314),
(10.47566706, 43.70637822), (10.47572377, 43.70635017),
(10.47566904, 43.70622286)]
#
polygon = [(10.47566407, 43.70653718), (10.47547900, 43.70648899),
(10.47552676, 43.70643001), (10.47572377, 43.70648684),
(10.47566407, 43.70653718)]
#
polygon = [(10.47529791, 43.70644799), (10.47543223, 43.70658753),
(10.47535860, 43.70659976), (10.47526109, 43.70647317),
(10.47529791, 43.70644799)]
#
#goal = [(10.47522229, 43.70654438)]
```

The following figure shows the initial situation of the mission. In addition to Zeno's initial position, we can see the red ball indicating the position of the goal and the green line representing the path Zeno plans to follow to reach the goal. The red dots, instead, indicate the points where the sensor beams encountered the obstacles (or, if no obstacle is encountered, the maximum range of the sonar).

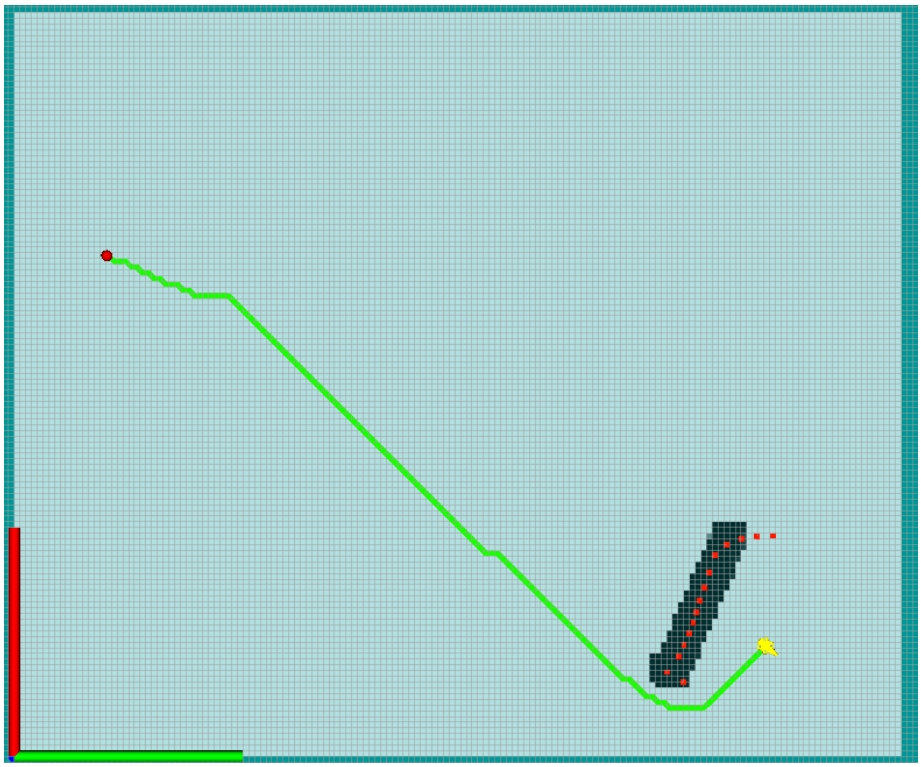

Figura 8: Zeno at the beginning of the mission

As the mission continued, while Zeno navigates around the map, the reconstructed map is evidently updating using the data obtained from the sensor. We can see the various shades of grey representing the confidence level of having an obstacle in a given cell.

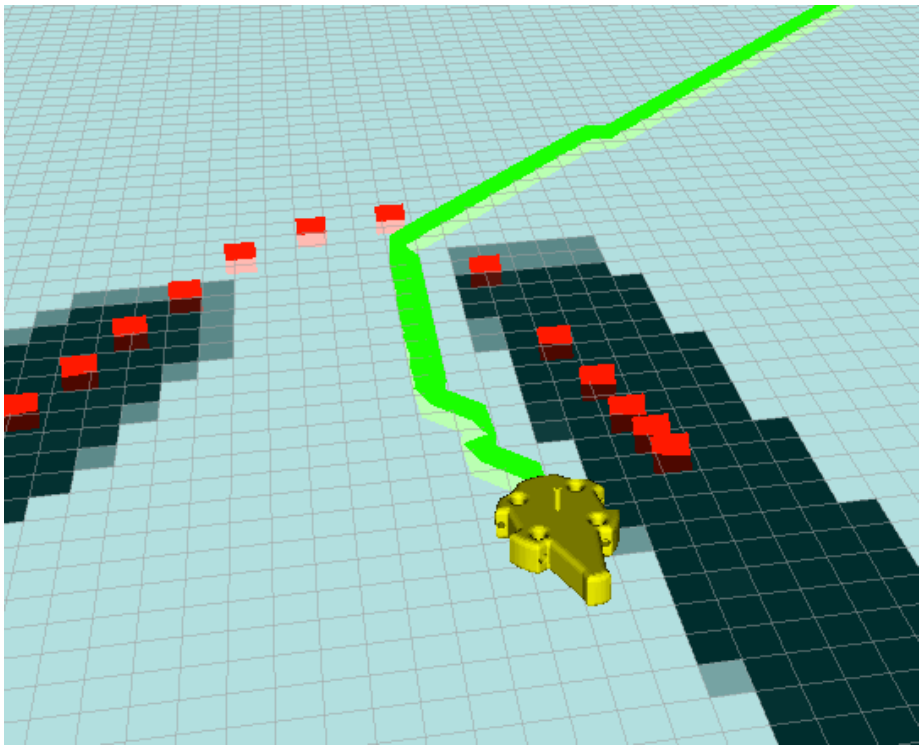

Figura 9: Zeno during the mission

As Zeno arrived to the goal position, we gained the partial reconstruction of the environment, only concerning the area where Zeno navigated.

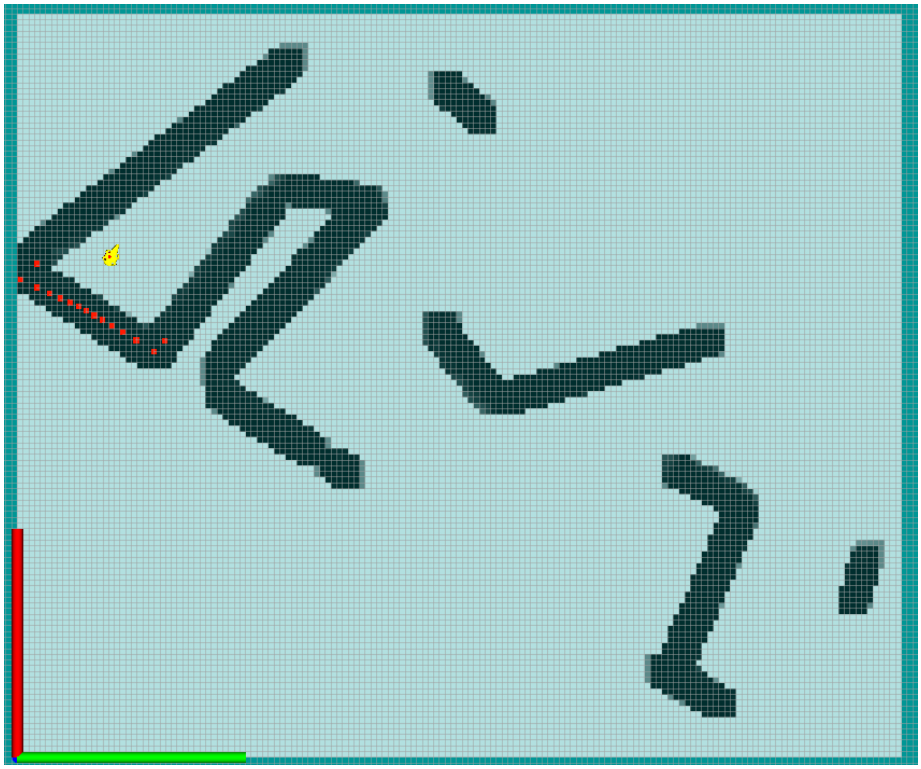

Figura 10: Zeno at the end of the mission

5.2 Scenario 2: Hard Scenario

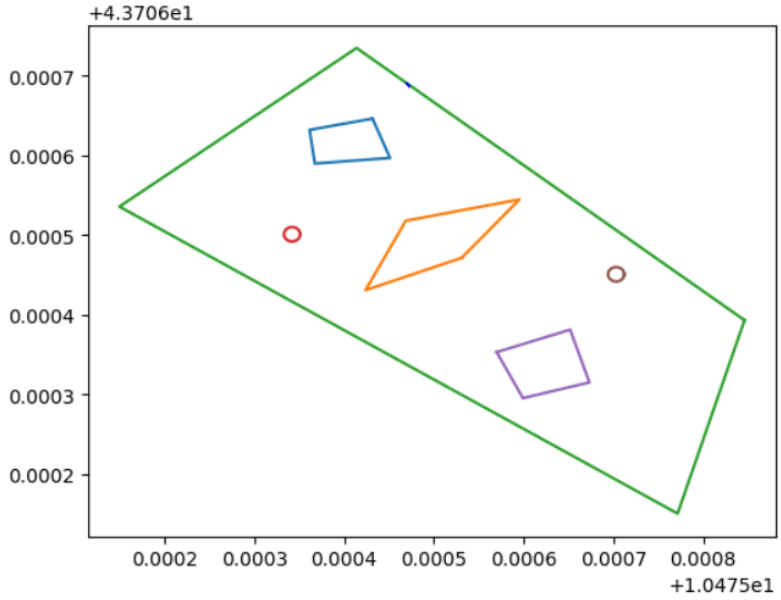

Figura 11: Representation of the map that was defined as a 'hard' scenario

Below are the coordinates of the map and the goal chosen for this scenario:

```
map = [(10.47584607, 43.70639344), (10.47541376, 43.70673506),
(10.47514949, 43.70653601), (10.47577151, 43.70615072),
(10.47584607, 43.70639344)]
#
polygon = [(10.47543155, 43.70664641), (10.47536139, 43.70663232),
(10.47536724, 43.70659005), (10.47545104, 43.70659710),
(10.47543155, 43.70664641)]
#
circle = [(10.47534190, 43.70650129), 1.0]
#
polygon = [(10.47553095, 43.70647171), (10.47559526, 43.70654497),
(10.47546858, 43.70651820), (10.47542370, 43.70643109),
(10.47553095, 43.70647171)]
#
circle = [(10.47570303, 43.70645107), 1.0]
#
polygon = [(10.47556993, 43.70635336), (10.47559916, 43.70629560),
(10.47567322, 43.70631532), (10.47565178, 43.70638154),
(10.47556993, 43.70635336)]
```

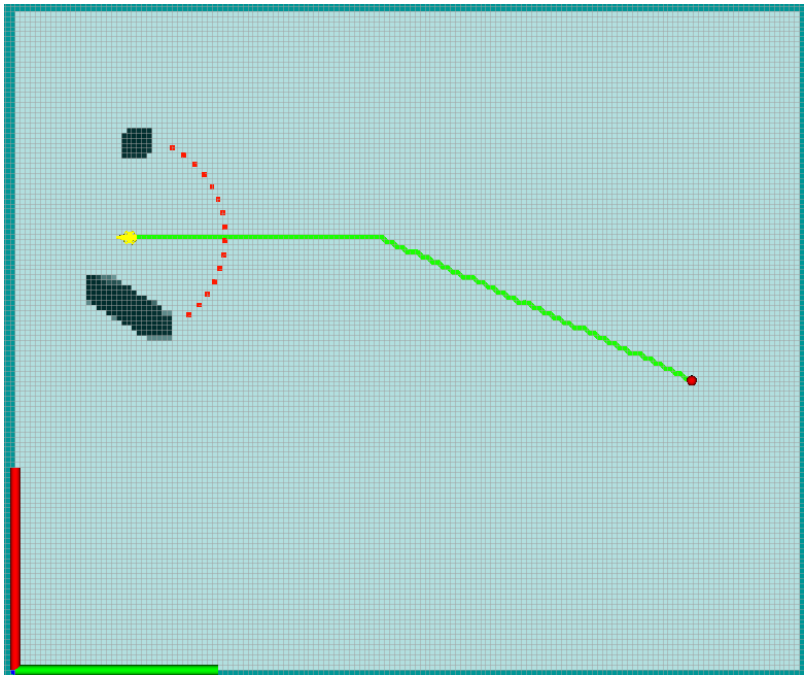

Figura 12: Zeno at the beginning of the mission

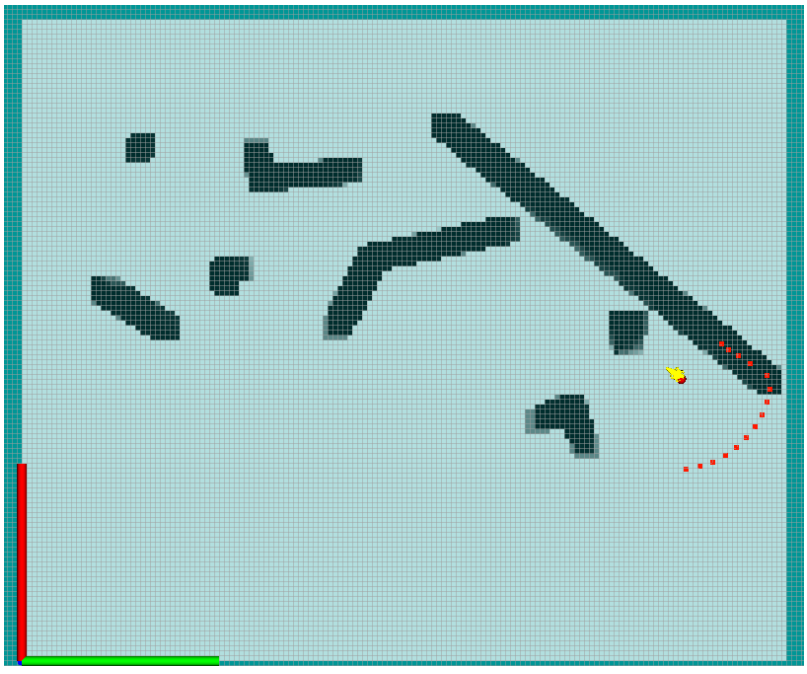

Figura 13: Zeno at the end of the mission

5.3 Extra

To better define the difference between the real obstacles and the safety margin, it was decided to do an additional simulation, slightly modifying the virtual map update algorithm. This modification allowed for better visualization graphically.

In this instance, the value of the cells within the margin are modified. For example, around the one where the obstacle was actually detected, the value is set to half the maximum value, and not increased by that amount. This way, the results are less conservative, as it is easier for those cells to become free again. Thereby, Zeno can get closer to the obstacles; however, the true shape of the obstacles appears clearer, distinguishing them from the safety margin artificially introduced by us.

The following figure shows what the map looks like if this change is implemented.

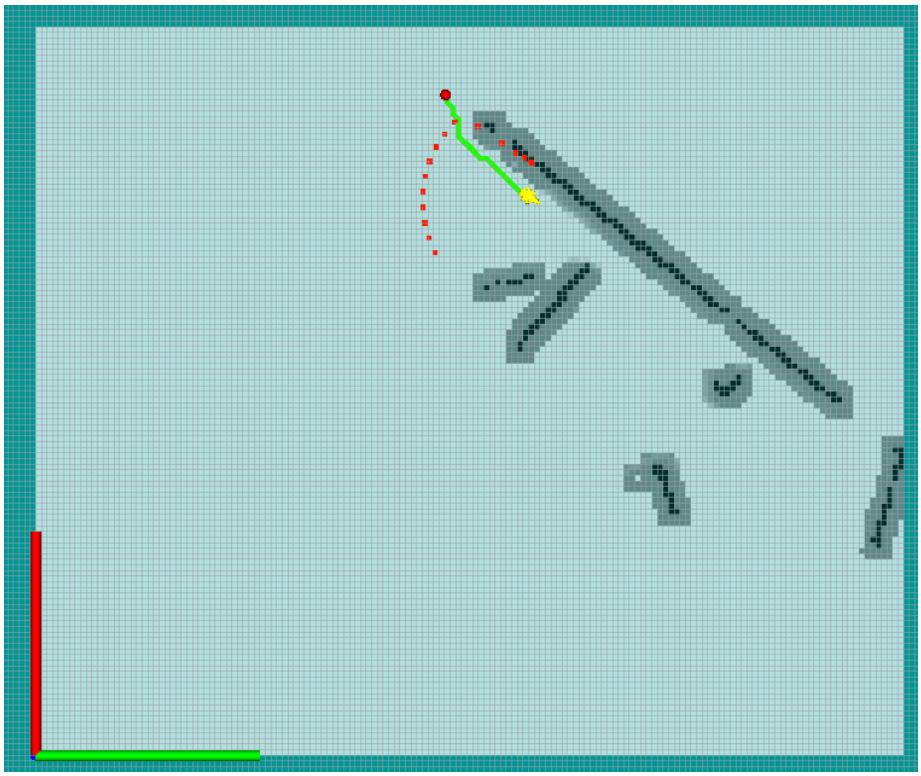

Figura 14: How the virtual map appears in this case

6 Conclusions

The experiment proved the algorithm to be capable of achieving the task with great effectiveness and safety, navigating to a given point through an environment full of obstacles successfully.
We believe this work is robust to changes of scenario, for example with larger maps, different obstacles and parameters values, higher resolution (smaller cells), and different management of the uncertainty of the obstacles.
It is more likely that additional efforts are needed to obtain similar results on a three-dimensional environment.