Group 01

1. Giuseppe Labate, giuseppe.labate.1@studenti.unipd.it
2. Greta Piai, greta.piai@studenti.unipd.it
3. Francesco Bordignon, francesco.bordignon.5@studenti.unipd.it

Repository link: https://bitbucket.org/prova123456/ir2324_group_01/src/master/

To clone the repository: git clone https://arcopigreta@bitbucket.org/prova123456/ir2324_group_01.git
Link to the video: IL LINKKKKKK!!!!

To run:

//In all terminals from now on:

>start_tiago

source ROS

Start the simulation and the navigation

In a new terminal start the server node:

>rosrun assignment1 server_rbt

In a new terminal start the server node:

>rosrun assignment1 client_usr x y z o_x o_y o_z o_w

Where: x,y,z are the coordinates for the positions and o_x o_y o_z o_w are the orientation's unit quaternion.

Code explanation:

The program is designed with a client/server architecture, more specifically with the client is callback based. The server receives the client's request to reach a point, moves the robot to the input position and prints out the number of obstacles and their associated coordinates in the map. More concretely, we developed a class ScanAction that implements the server functions; in the constructor, we instantiate a subscriber for each of the topics we will need to execute the various operations: one for interacting with the robot's status, one for dealing with the map properties to establish if a certain point belongs to an obstacle or the map itself, one for receiving the input from the scan sensor and one for controlling the odometry. statusCallback reads the topic "move_base_simple/status" to send to the client the current status of the robot, whether it accepted the goal and it is, then, moving, it reached the goal or it started or finished the detection of the obstacles. odomCallback simply retrieves and converts the coordinates in the appropriate reference system. scannerCallback does the obstacle detection only when the robot reaches the final goal. It clusterizes points near to each other into the same cluster. Private method columns_from_clusters defines column objects that are sufficiently far from the wall, to test please consider not to put the obstacles to be detected not too close to a wall. Private method toEuclidean converts in Euclidean coordinates the position's values. mapCallback saves the map into the program for further use. executeS gets the input goal from the client and moves the robot to the final destination, paying attention to the given map's properties. We considered as "wall" every object that was too close (considering a certain threshold) to a wall defined on the map and all the objects that were not "similar" enough to a column (we know a column is not as wide as a drawer) to simplify the detection.

Greta took care of writing the client, the server's structure, and this report.

Giuseppe dealt with moving the robot, making sure that the client got the right feedback on the robot's current status.

Francesco worked on the detection phase.

We all worked to merge the different parts of code.