Intelligent Robotics - Assignment 1
Group 09

Università degli Studi di Padova
DEI - Department of Information Engineering
Academic year: 2023/24

Andrea Campagnol
andrea.campagnol.1@studenti.unipd.it
Daniele Fabris
daniele.fabris.5@studenti.unipd.it
Arash Abdolhossein Pour Malekshash
arash.abdolhosseinpourmalekshah@studenti.unipd.it

Link to the repository
Link to the additional material

1 Introduction

1.1 Purpose

The main purpose of the assignment is to implement a routine that allows Tiago to navigate in the given environment, in order to reach a final pose. Once the robot is in its final pose, it detects the movable obstacles and print on the screen their respective position.
Our implementation is based on a simple Client/Server structure, where the action client receives the final pose from the user through the command line and calls the action server which executes all the tasks. As a result the action server sends the final list of the obstacles' positions back to the client. The action client also implements callbacks to the feedback of the action in order to receive information while the server is carrying out tasks.

1.2 Brief explanation

Motion Control Law

At the beginning the robot is in its initial pose and the first task to carry out is to cross the narrow corridor. We have implemented a control law working with laser scanner data received from the */scan* topic, without calling the *move_base* stack that we know it to be inefficient in narrow passages, but directly sending the velocity commands to the */cmd_vel* topic to move the robot.

Autonomous moving

Once Tiago has reached the end of the narrow corridor, starts to move to the desired final pose using the navigation stack. In the absence of dynamic obstacles, the *move_base* node will eventually get within a tolerance of its goal or signal failure to the user.

Obstacles detection

We have implemented two methods in order to detect the obstacles. The first one is based on the analysis of the laser scanner data, grouping together close points and checking if these sets could actually be cylindrical objects, while the second one converts laser scanner data into a distance map that is processed, separating the background (walls and shelves) and objects in order to detect obstacles.

2 Strategies and Algorithms

2.1 Controlled moving

In order to make Tiago cross the narrow corridor, some fundamental controls must be taken into account:

- At the beginning we must check that the robot is in the initial pose, otherwise we cannot execute our controlled motion control law and we directly impose to the robot to move using the ROS navigation stack.
- Velocity in free space is broken into its linear and angular part and each of them must be constant over time.
- While crossing the corridor, we must continuously check the data received from the laser scanner in order to detect obstacles within a certain range in front of the robot, check if Tiago has reached the end of the corridor or if the robot is still going through the narrow corridor it's important to monitor if it's going straight or if it's getting to close to a wall.
- During the correction of its orientation, it's good practice to not use an high linear velocity and to keep low the angular speed to for precision reasons, but also to avoid getting too close to any obstacles while correcting the pose. Obviously if the robot is closer to the wall on the left, we must rotate it to the right until we reach a satisfactory orientation, while if it's closer to the right wall, the rotation must be done to the left.
- Every time we modify Tiago's velocity, even in the case of stopping completely, we publish it on the */mobile_base_controller/cmd_vel* topic.

As we know, subscribing to the */scan* topic we receive a *sensor_msgs/LaserScan* message containing all the information about the Tiago laser scan. In order to find the indexes of frontal, left and right range values, we have done a simple computation based on knowledge of the start and end angles of the scan (*angle_min* and *angle_max* respectively), the angular distance between measurements (*angle_increment*) and the number of range values.

$$angle\_max \cdot \frac{180°}{\pi} = 110°$$
$$90° = 110° - 20°$$
$$20° \cdot \frac{\pi}{180°} = rad\_20$$
$$rad\_20/angle\_increment = index\_90 = 60$$

where *rad_20* is the value of 20° in radiant and 60 correspond to the index of range scan pointing to the right wall, that is the one at 90 degrees clockwise from the front view index. Then, we can simply derive the left and frontal indexes.
Each time we need to update the values relative to the distances in the three direction, we call the callback function *scanCb* to retrieve scan data and the other to do actually update the parameters /textitupdateParameters.

2.2 Autonomous moving

Once the robot has reached the end of the narrow corridor, the robot switches to autonomous movement through the use of ROS navigation stack.
The *move_base* action client sends the final pose as a goal to the action server and wait for the result, that is if the robot has reached the final pose or has failed to.

2.3 Obstacles Detection

Range Detection

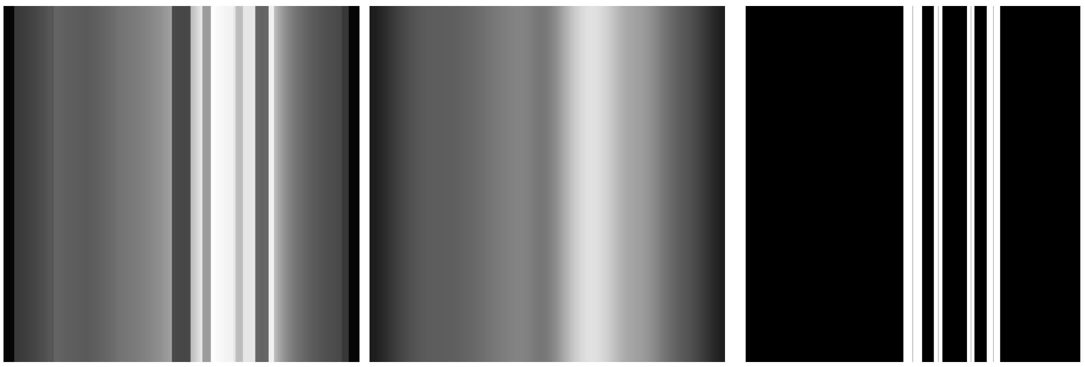

Figure 1: Distance Map - Background - Obstacles with Midpoints

The first approach in order to detect the obstacles, once the robot has reached its final pose, consists of working directly on the laser scanner data. First of all, we transform the scan into points in the coordinate system of the laser sensor. Then, we create set of points formed by a sufficient number of close points. Then we must check if these sets of points could actually be cylindrical obstacles (circles in the 2D plane), and the way we did this check is based on the computation of the center and the radius of a circle given three points, the first, the middle and the last in the respective set of points. Obviously, we consider circles with a radius under a certain threshold, because we know the cylindrical object we're going to detect.
Furthermore, a problem to consider is that the laser sensor is located in front of the robot and in its scans it detects parts of the robot's body, so we must eliminate unnecessary data. This last point makes us understand that, in the end, we need to apply a transform to change the frame from that of the sensor to the *base_link*. At the end, the central points of each detected circle will be returned.

Distance Map Detection

The second approach, like the first one, uses the data provided by the laser scanner. However, it does not use them directly to detect obstacles, but uses them to generate a distance map. A processing phase is then carried out to detect obstacles on the generated image. The underlying idea is very simple and is based on the assumption that the obstacles inside the room are well separated objects and they are very small compared to the background objects (walls and shelves). We first compute the euclidean distance between the laser sensor and each detected point in the space, we normalize it and convert it into a grayscale (0-255) image. We can notice that areas of the distance map related to the background points, have a homogeneous gradient color: neighbor pixels differ slightly. Instead, obstacles areas pop out from the background. Assuming that the obstacles are much smaller in size than the background, if we compute a strong Gaussian blur filter on the distance map, we obtain a new image that approximates the real background, because obstacles are faded into the background (Figure 1). Obstacles are than detected by computing the difference between the distance map and the background image: only areas related to obstacles lead to high difference values. Two thresholds are used to validate the detection: only difference values greater than 10 are taken into account and only objects larger than 3 pxs are considered obstacles. By doing so, false detection caused by edges or slits should be avoided (A future improvement could be done by adjusting these thresholds, e.g. with an automatic/dynamic threshold selection). The resulting image has white stripes in correspondence of the obstacles, and it's easy to compute the middle point of each stripe. Since the image has the same width in pixels as the number of laser scans, we can easily convert these midpoints to the related points in the space by looking at the column index of the pixel.

The two approaches will have small differences in the coordinates, because in the first one it is computed the center of the cylindrical object, while in the second one we take the middle point in the gray scale image section of the object, corresponding to a point in the circumference of the obstacle.