UNIVERSITY OF PADUA

INFORMATION ENGINEERING DEPARTMENT (DEI)

MASTER'S DEGREE IN COMPUTER ENGINEERING

Report Assignment 1

Prof.
Emanuele Menegatti

Students:
Francesco Caldivezzi : 2037893
Manuel Barusco : 2053083
Riccardo Rampon : 2052416

Contents

1 Introduction

In this report we are going to explain the main details of how our solution for this assignment works. In particular we are going to explain:

- the code structure of our solution.
- how we have solved the problem of obstacles detection, which includes walls detection and circular movable obstacles detection that are both comprises in the environment.
- how we have solved the problem of the navigation through the narrow corridor using the laser data.

2 Code Structure

The structure of our code is based on a simple action client-server architecture where the client sends the goal to the server and the server does all the tasks. The main components that we developed are:

- the Action Message (MoveDetectAction)
- the Action Client (TiagoClient)
- the Action Server (TiagoServer)

2.1 Action Message (MoveDetectAction)

We developed a custom action message with the following components:

- Goal: field composed by three float64 variables that are the x,y coordinates of the final pose of the robot and the orientation w.r.t. the Z axis of the robot. All these three coordinates are based on the map reference frame.
- Feedback: a uint8 variable with the following meaning for every value: 0 if the robot is moving, 1 if the robot is arrived to the final pose, 2 if is scanning, -1 if an error occurred in the navigation, -2 if an error occurred in the detection.
- Result: a geometry_msgs/PoseArray variable with the coordinates w.r.t the Robot reference frame of the detected obstacles in the final pose.

2.2 Action Client (TiagoClient)

We developed a custom action client that simply sent the goal (coordinates of the final pose) to the robot. The coordinates that are sent are passed to the client from the user by using the command line. The client then received updates from the TiagoServer about the status of the task.

2.3 Action Server (TiagoServer)

We developed a custom action server that managed all the tasks and that controls the Robot. This server is actually a proxy because it contains an action client that is used to communicate to the move_base action server when it is needed. In fact this server has the following behaviour:

- It receives the goal from the TiagoClient and it updates it with the MoveDetectAction Feeback messages.
- It moves the Robot to the final position by using the navigation stack (First task of the assigment) or by using our Narrow Passage control law for moving in the corridor and then the navigation stack when the robot is outside the corridor.
- It does the detection of the moving obstacles when the robot reaches the final pose.

3 Detection

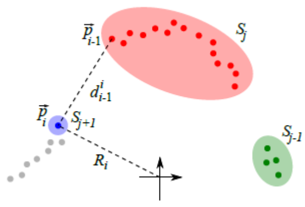
The detection task is the task that the robot will execute whenever it reaches its goal position. During such a task the robot needs to identify the position of the circular tables inside the map. To solve this task, we used the data coming from the laser sensor (topic /scan). So, given those data we develop an algorithm that can be summarized with the following steps :

- classify points detected by the laser into two different classes :
  - circular tables points
  - not circular table points (i.e. points of walls, bookshelves etc..)

3.1 Classification of the points

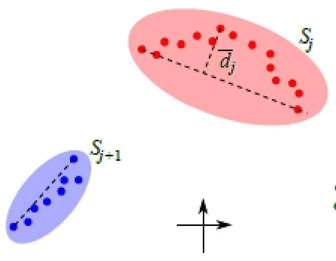
The first step of the detection algorithm was developed by adapting an obstacle extraction [1] algorithm that is able to find straight lines and circles. In particular, this algorithm consists of :

- Grouping points : this step consists of grouping points in order to provide a collection of subsets S_j representing possibly separate objects. A point $\vec{p}_i$ is assigned to the group of point $\vec{p}_{i-1}$ if the following criterion is satisfied : $d^i_{i-1} < d_{group} + R_i d_p$ where :
  - d^i_{i-1} is the euclidean distance between point $\vec{p}_i$ and $\vec{p}_{i-1}$
  - d_{group} is a user defined threshold for grouping
  - R_i is the length of the vector formed by connecting the origin and point $\vec{p}_i$
  - d_p is the angle of increment of the laser sensor

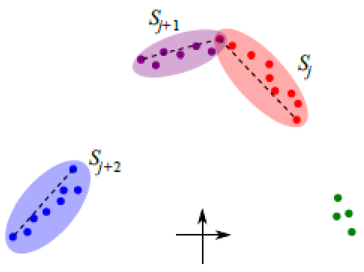


- Splitting points : in this phase each subset S_j of points is examined only if it contains a number of points greater than N_{min} points. Moreover, in this step, an Iterative End Point Fit algorithm is used in order to build a leading line upon two extremes points of the group and seeks the point of the group, farthest apart from this line, that is used for splitting the subset. The criterion of splitting two sets in two is the following : $\bar{d}_j > d_{split} + R_j d_p$ where :
  - $\bar{d}_j$ is the distance between the leading line and the farthest point of set S_j
  - d_{split} is a user defined threshold for splitting
  - R_j is the length of the vector formed by connecting the origin and the farthest point in S_j
  - d_p is the angle of increment of the laser sensor

  Notice that this procedure is repeated recursively for each new subset until no more splitting occurs. Furthermore, in case of splitting, the point of division is assigned to both the new subsets.

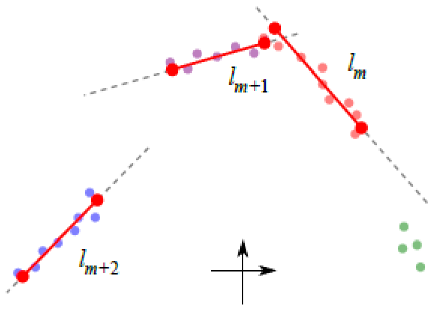


Before Splitting

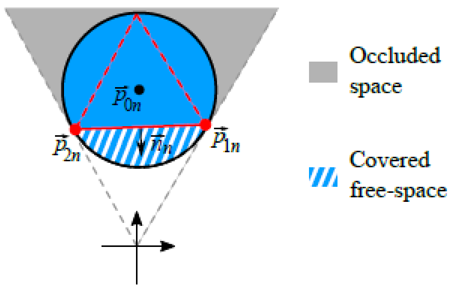


After Splitting

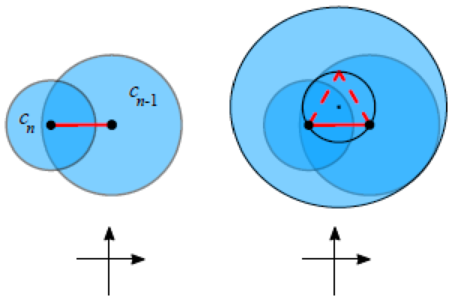
- Segmentation : In this step given a subset S_m of points, it is approximated with a segment. This segment is built by using this equation : $A_m x + B_m y + C_m = 0$ which is fitted by using least square regression to compute the parameters.
- Segments merge : in this phase, given the collection of extracted segments, they are subject to a merging procedure that is composed of two parts :
  - connectivity test : checks if both segments are close to each other by comparing $d_0 < d_{merge}$ where :
    - d_0 denotes the distance between neighboring points of both segments
    - d_{merge} is a user defined threshold for connectivity
  - spread test : checks if both segments are collinear with the following test : $max(d_1, d_2, d_3, d_4) < d_{spread}$ where :
    - d_{1-4} are distances between segments extreme points and the new leading line
    - d_{spread} is a user defined threshold for spread



- Circles extraction : in this step, given the set of previous segments detected, we construct for each one of them a equilateral triangle with base equal to the segment and central point away from the origin. Next, on the basis of the triangle we construct an circumscribed circle c_n with radius : $r_n = \frac{\sqrt{3}}{3}\bar{l}_n$ ($\bar{l}_n$ is the length of the segment) and central point : $\vec{p}_{0n} = \frac{1}{2}(\vec{p}_{1n} + \vec{p}_{2n} - \vec{p}_{3n})$. Finally, the radius is enlarged by a user defined threshold r_d and if the resulting radius after enlargement is not greater than another user defined threshold r_{max} then the circle is added to the set of circles.



- Circles merge : in this phase, given the circles detected in the previous phase, is subject to a merging procedure in order to merge two circle that intersect each other and are very close.



So, after adapting the code, we were able to find lines and circles given the data provided with the Laser Range Finder.
Finally, we return the center of the circles as final position of the objects. Notice that, points are computed by rotating the axis as shown in the **Narrow Passages** section, but, at the end we return the position of the centers of the circular tables in the original reference frame of the robot (not rotated).

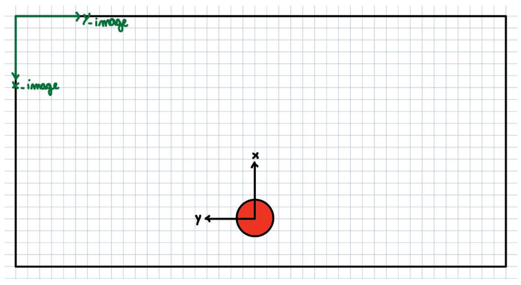
4 Narrow Passages

The Navigation in the Narrow Space task is the task that allows the Tiago robot to move autonomously in a narrow space without needing to use the ROS Navigation Stack, which involves the move_base stack, which is inefficient for the navigation in narrow spaces. We implemented a method that is divided into three parts:

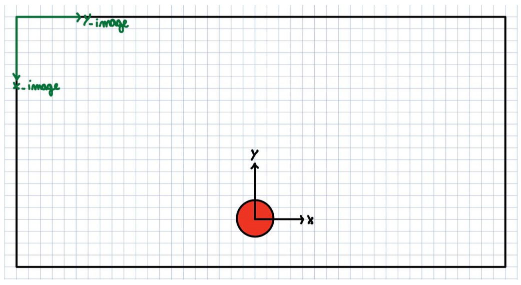
- in a first one we have the construction of a binary image from the data provided by listening to the topic $'/scan'$;
- then on the obtained image we applied some Morphological operators that help in the detection of lines using the Hough Transform;
- finally we used two methods to refine the lines and clustered them into two distinct clusters representing the walls of the corridor in front of the Tiago robot.

4.1 Construction of the image

The first step of the Navigation in a Narrow Space task is the construction of an image from the data provided by reading the *sensor_msgs/LaserScan* message, subscribing to the topic $'/scan'$. In order to construct such an image, some formulas are required to change from angles in radians and distances in meters to true integer x,y coordinates. First, we rotated the reference system of the Tiago robot by $\pi/2$ so that the y-axis and x-axis are oriented correctly, as shown in the figure below:



Old reference frame



New reference frame

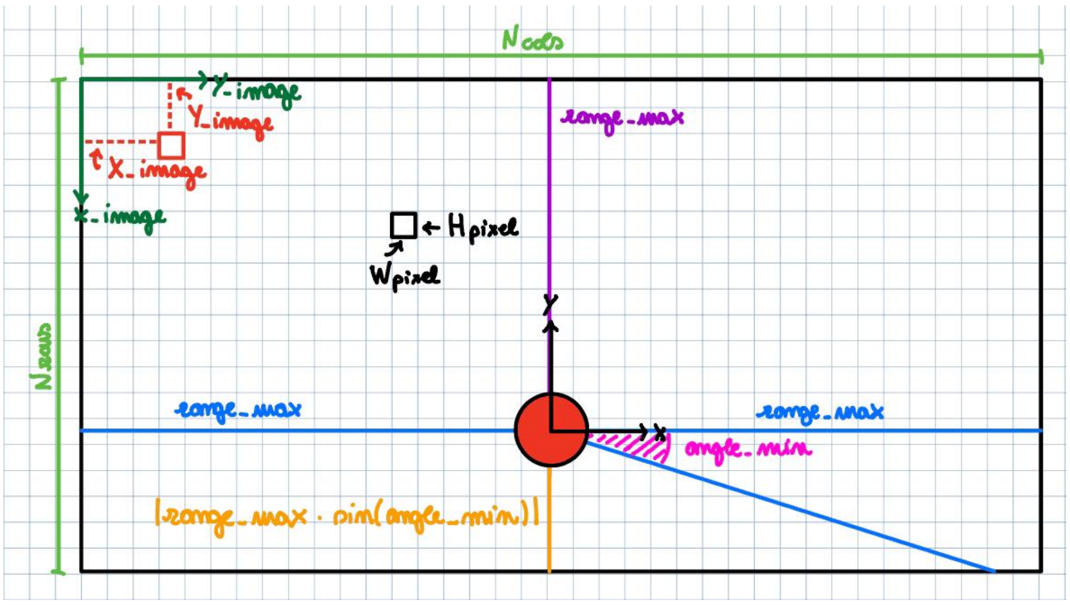
Next, we calculated useful quantities to start processing the laser scan data and then obtain integer coordinates in x and y; the formulas used are as follows:

- $N_{Rows} = \frac{(angleMin+\pi/2)}{angleIncrement}$, to get the number of rows of the image that represent the scan of the laser;
- $N_{Cols} = \left\lfloor \frac{\pi}{angleIncrement} \right\rfloor$, to get the number of columns of the image that represent the scan of the laser;
- $W_{Pixel} = \frac{2*rangeMax}{N_{Cols}}$, to get the width of a pixel of the image that represent the scan of the laser;
- $H_{Pixel} = \frac{rangeMax+(rangeMax*\sin(angleMin))}{N_{Rows}}$, to get the height of a pixel of the image that represent the scan of the laser;
- $X_{RawImage} = rangeMax + \rho * \sin(\theta)$, to get the 'x' float coordinate in the image reference frame of a point detected by the laser;
- $Y_{RawImage} = rangeMax + \rho * \cos(\theta)$, to get the 'y' float coordinate in the image reference frame of a point detected by the laser;
- $X_{Image} = \left\lfloor \frac{XRawImage}{H_{Pixel}} \right\rfloor$, to get the 'x' coordinate in the image reference frame of a point detected by the laser;
- $Y_{Image} = \left\lfloor \frac{YRawImage}{W_{Pixel}} \right\rfloor$, to get the 'y' coordinate in the image reference frame of a point detected by the laser;

where

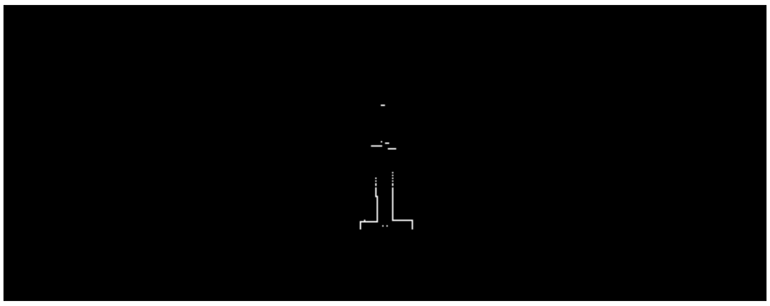
- $angleMin = angle\_min + \pi/2$ with $angle\_min$ that is a field of $sensor\_msgs/LaserScan$ message, which represent the start angle of the scan [rad];
- angleIncrement: is the field $angle\_increment$ of $sensor\_msgs/LaserScan$ message, which represent the angular distance between measurements [rad];
- rangeMax: is the field $range\_max$ of $sensor\_msgs/LaserScan$ message, which represent the maximum range value [m];
- ρ : is the distance obtained from reading each i-th scan from $sensor\_msgs/LaserScan$ message;
- θ : is the angle obtained from $angleMin + (angleIncrement * (i - th\_scan))$;

In order to have a visual proof of the quantities listed above, we show the following image that summarises them graphically:



Visual proof of the quantities listed above

The image is then constructed in such a way that each scan provided by the *sensor_msgs/LaserScan* message corresponds to a white pixel in a completely black image. The result obtained is as follows:



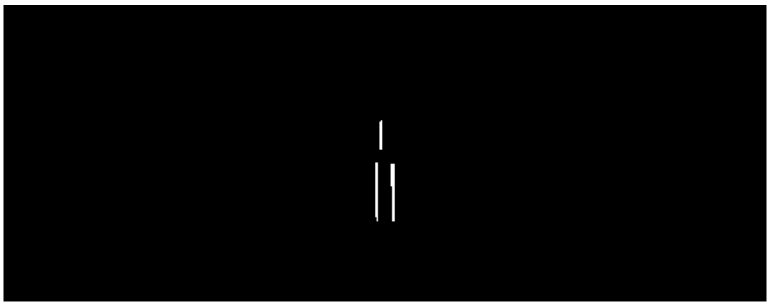
The Built Image

4.2 Morphological operators and Hough Transform

The second step of the Navigation in the Narrow Space task requires the application of morphological operators such as dilation and erosion because they allow the application of a structuring element to the input image in order to shrinks or to expands the image pixels. In this case, we applied a dilation with a rect structuring element of parameters (2,25) to the input image and then an erosion with an equal structuring element of parameters (1,35) in order to shrinks better the image pixels. After this post-processing of the image pixels, we applied the probabilistic Hough transform with this parameters:

- rho = 1;
- theta = $\pi/180$;
- threshold = 1;
- minLineLength = 8.0;

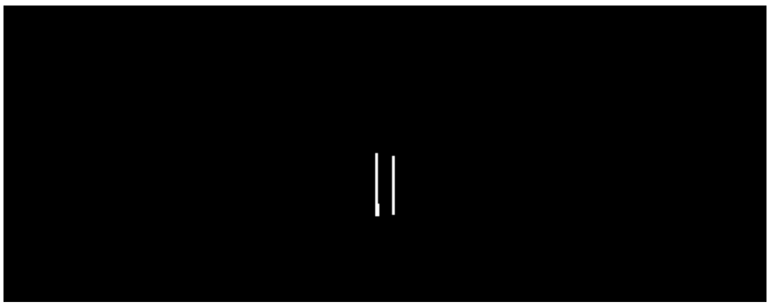
The result image obtained is the following:



The Image After Hough Transform

4.3 Refining and clustering the lines

The third and last step of the Navigation in the Narrow Space task involves the use of a first clustering method to divide the detected lines into two distinct groups; this method is OpenCV's 'partition' function, which requires the specification of a predicate to perform the division; we chose a predicate that predicted that a line would belong to the cluster if it was less than a certain distance that we specified as a threshold. After this step we created a function that merged the lines belonging to the same cluster into two distinct lines, one per cluster, based on the average of the x,y coordinates of the points defining the single line, and then another function that refined them by considering the line with the major y coordinates, and stretching the one that was minor by looking to y coordinates, and viceversa. This is our final result:



The Final Image

5 References

[1] Mateusz Przybyla. Detection and tracking of 2d geometric obstacles from lrf data. pages 135–141, 07 2017.