Lab2 Report - Line features with Split & Merge

Mohammad Rami Koujan

1 Introduction

The main focus of this lab is on how to extract features from data points. Since precise position estimation plays an important role in any navigation system, e.g. path planning, localization, and dynamic map building, it is very important to define this position accurately. In fact, data acquired from odometry provides an undetermined position error. Therefor, it is essential to use some extroceptive sensors, such as laser rangfinder, in order to provide more precise measurements. Laser sensor, indeed, provides 2D data which can be used for feature extraction and consequently in one of the previously mentioned applications. Basically, the main task in this lab assignment is to implement the *Iterative-End-Point-Fit* algorithm which is a split and merge method.

2 Algorithm analysis and implementation

Split-and-Merge method is one of the most well known algorithms. It works in the following way:

- starts from a set of points, in our case it is the laser rangefinder data which is stored in a bagfile dataset.
- fits a line to that given data set.
- finds the points that have the maximum distance from the fitted line.
- checks whether this distance is smaller than a threshold or not. if yes, it moves to another data set. Otherwise, it splits the line at the found point and inserts those two lines instead of the original one in the set.
- finally, when all the sets have been tested, it merges collinear segments.

The method of implementation is mainly based on two nodes. One for handling the odometry of the robot the other one is for dealing with the laser scan data. Concerning the splitandmerge file, it contains two main functions, called inside another function. The first one is used for performing the split step and the second one for doing the merging part.

The split function takes the set of points, split threshold, inter-point distance threshold, minimum number of points to be considered as a set, first point, and finally last point as input arguments. It also gives back an array of points that contains the first and last points of each split line. The procedure starts, after checking the number of passed points if it is greater than the minimum number or not, by calculating the parameters of the line that passes through the first and last point of the given set. These parameters are calculated mathematically as follow:

$$a = y1 - y2 \tag{1}$$
$$b = x2 - x1 \tag{2}$$
$$c = x1 * y2 - x2 * y1 \tag{3}$$

After that, the distance between each point in the points array and the previously computed line is calculated using the following formula:

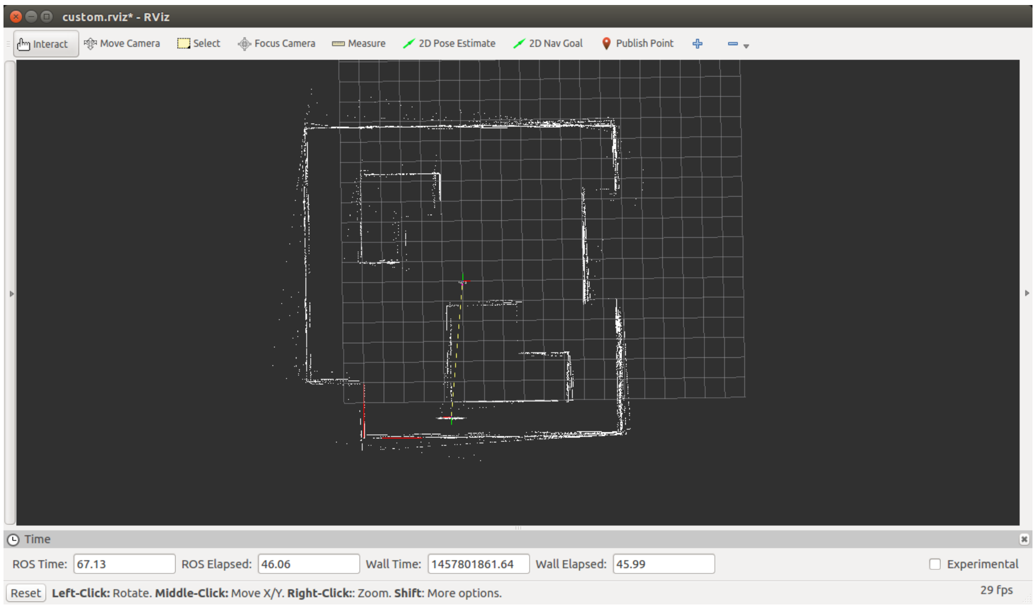
$$D = \frac{|ax + by + c|}{\sqrt{a^2 + b^2}} \tag{4}$$

The only point with the highest distance from that line will be checked to ensure that this distance is not greater than the distance threshold.

If it is greater, the set of points will be split into two sublines. The first one is passed again to the split function, recursively, with `first_point` parameter that has the same as the previous value and last point that has the same as the point of split that has just detected. For the post part of the splitting process, it will be also passed with `first_point` as the detected point plus one and `last_point` as the current value. Subsequently, the prev and post parameter will be checked ,after the end of the recursive callback, whether they have a None value, some points, which means they will be stacked together, or only one of them has points to be returned.

Nevertheless, if the maximum distance is less than the given threshold, the inter-point threshold will be checked this time. The result of this check will have the same as the previous procedure, either splitting with recursive call or not.

Apropos the merging function, it merges two lines if they are collinear and have an angle difference and distance difference that are smaller than the given thresholds. Otherwise, it does not change anything. The following figure presents visually the results of testing the explained split and merge algorithm on the shown environment

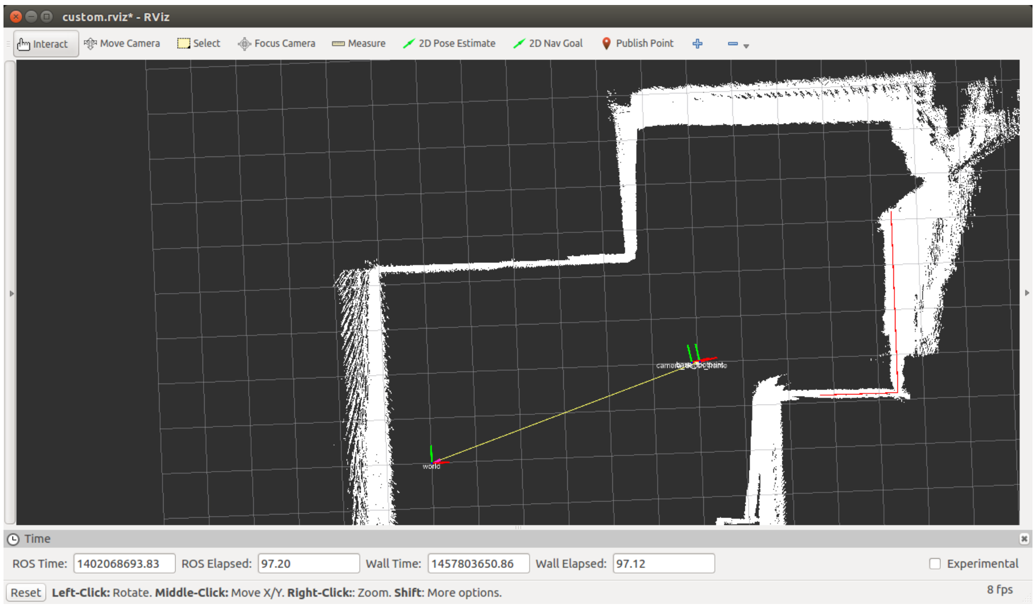


ROS Time: 67.13 ROS Elapsed: 46.06 Wall Time: 1457801861.64 Wall Elapsed: 45.99

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:**: Zoom. **Shift:** More options.

Figure 1: Testing of split and merge algorithm with default parameters on dataset3.

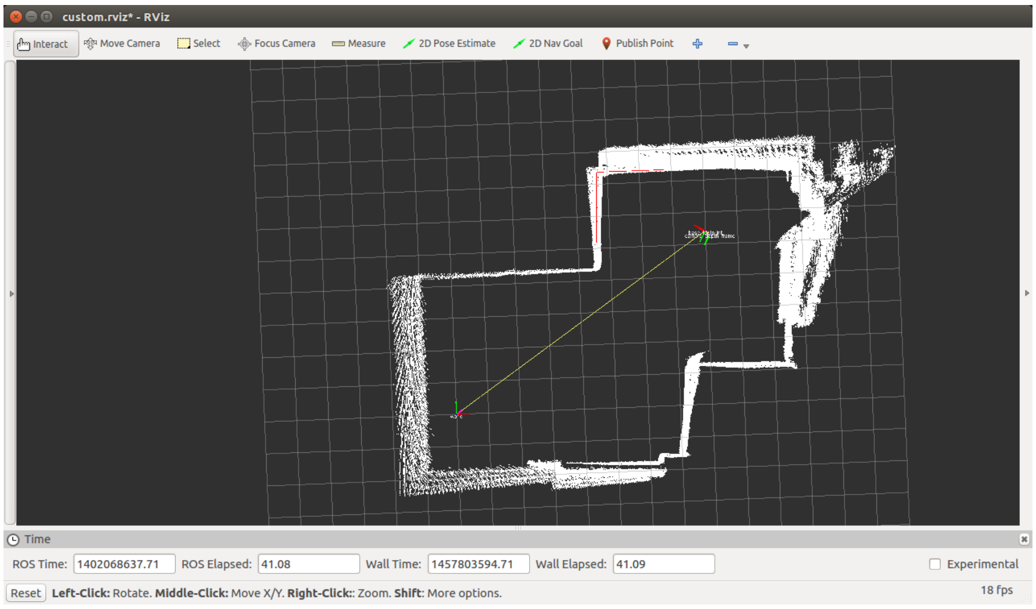
For higher value of distance threshold and inter-point threshold the algorithm start splitting from a higher distance as opposed to the current case with default parameters. The next figure also shows a snapshot of the test on dataset2.



ROS Time: 1402068693.83 ROS Elapsed: 97.20 Wall Time: 1457803650.86 Wall Elapsed: 97.12

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:**: Zoom. **Shift:** More options. 8 fps

Figure 2: Testing of split and merge algorithm with default parameters on dataset2.

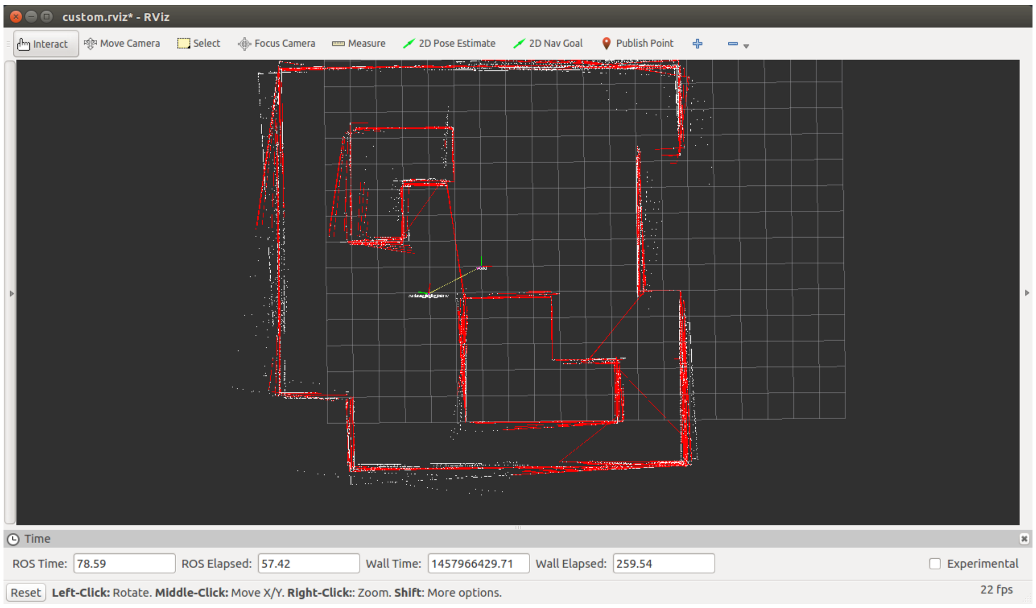


ROS Time: 1402068637.71 ROS Elapsed: 41.08 Wall Time: 1457803594.71 Wall Elapsed: 41.09

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:**: Zoom. **Shift:** More options. 18 fps

Figure 3: Testing of split and merge algorithm with lower value of merging threshold on dataset2

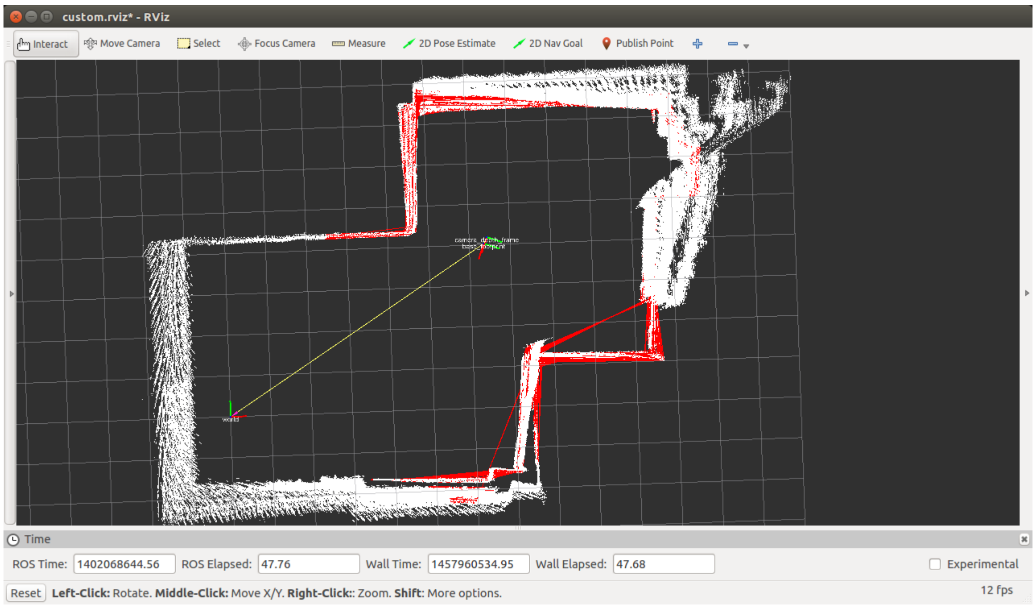
In order to represent all the lines that have been obtained by the Split and Merge algorithm in a single image, the `marker_id` parameter should be increased by one and then published each time a LaserScan message with topic ”scan” arrives. Figure 4 shows all the lines obtained on one map for the two different data sets.



ROS Time: 78.59 ROS Elapsed: 57.42 Wall Time: 1457966429.71 Wall Elapsed: 259.54

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:**: Zoom. **Shift:** More options. 22 fps

Figure 4: Showing all the obtained lines for the map of dataset3.



ROS Time: 1402068644.56 ROS Elapsed: 47.76 Wall Time: 1457960534.95 Wall Elapsed: 47.68

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:**: Zoom. **Shift:** More options. 12 fps

Figure 5: Showing all the obtained lines for the map of dataset2

3 Conclusion

In this lab session, a spilt and merge algorithm is implemented successfuly using python language. It is also tested with two different dataset files registered from different environments. Inspecting the results signifies that odometry informtion alone is not sufficient for giving precise details about the robot position which necessitates the use of other sensors, e.g. laser range finder with the aid of feature extraction to provide better results.