PROBABILISTIC ROBOTICS

Lab report 3: Line Features with Split and Merge

Author: **RAABID HUSSAIN**

Objective:-

It is usually inevitable in mobile robotics for the robot to know about its environment and where it is at the moment in that environment. This helps the robot to better decide what action to take next. Since odometry sensors are not sufficient to provide this information, a need for rangefinder and vision sensors arose. The rangefinders often provide with dense information making them popular.

The basis of any mapping algorithm is to extract features from the environment. A common feature is to extract information regarding obstacles in form of lines representing their outer surfaces. This lab work was dedicated to one such technique called the split and merge algorithm. Iterative end point fit scheme was used to define outer boundaries of the obstacles in a map. The main objective of the lab-work was to extract line features of data point cloud and to match them to draw an appropriate map of the environment.

Methodology:-

This lab work was to implement a split and merge line feature extraction technique on ROS. Since feature extraction is the basis of many localization and mapping algorithms, this algorithms to be used in all the coming labs. The algorithm was tested on ROS simulation platform, in which a robot traversed through a maze making a map of the environment as it passed obstacles.

A research paper titled ‘A comparison of line extraction algorithm using 2D laser rangefinder for indoor mobile robotics’ was thoroughly studied as it gave an overview of all the algorithms developed and provided us with a brief yet comprehensive overview of the split and merge algorithm. The iterative end point fit algorithm was used. A skeleton of the python code was provided to us and we were to implement the algorithm using that skeleton.

The first thing we did was to run the program in its default mode, which made a rough map of the environment and the lines extracted from the dataset were, in most of the cases, not on the walls. Rather the lines ran from one end and straight to the other end, not taking into account the shape of the wall in between.

The split and merge algorithm works under the principle to fit the best possible line through a given set of point clouds. An initial line drawn from two points is compared with all the points in the dataset to decide where to split the line. If the line passes through the test, it is not split otherwise the line is divided into two parts at the position of the point that gives the worst result. The split lines are then passed through a merging test to decide if the split lines have the same geometry in space and can be considered one or not.

The process is repeated on all the data clouds to provide an appropriate geometry obtained from the point clouds.

For our implementation, a previous recorded data was provided in a bag file and we were to extract line features from that data cloud. The data points for every instance of time was read separately from the file and a line approximated from those points is drawn in red on the computer screen.

The algorithm starts by taking the end points of the instance into account and approximating a line joining them. The equation of this line is then used to calculate the distance of every point, in the instance's point cloud, form this line. The point with the maximum distance to that line is saved and its distance compared to a threshold. If the distance falls out of the threshold zone, then the line is split at that point forming two lines: from the start point to the maximum distance point and from that point to the last point. These new lines are again fed into the system until no more splitting is required.

I initially tried to match the distance of every point with the splitting threshold. This however, resulted in wrong results as the splitting was wrongly being implemented. The problem had risen due to a lack of understanding of the entire code structure. However, after a thorough examining of the entire code and deep learning of the algorithm, the mistake was identified and cleared.

If the line passed through the test then an additional check is applied that checks if all the consecutive points are close to each other indicating whether all points are of the same obstacle or different. If two points are not close to each other, the line is again split at this point and the above process repeated. Otherwise the algorithm proceeds onto the merging step.

In order to merge any two lines, it was necessary to check if the split lines indeed belonged to the same line segment. For this, two checks were applied: firstly that the start point of the second line is very close to the ending point of the first line and secondly that the angles of the two lines are almost the same with respect to the reference world system. Only if both of these conditions are met then the lines are merged otherwise the line are kept separate. All the lines are gone through this check to obtain a final line.

Here a problem I faced was that initially I was deleting separate points instead of the line segments (problem caused due to less familiarization of the python syntax). This led to the program being crashed at the next iteration after it had tried to merge two lines.

The merging step is important as it reduces the number of line segments produced by the algorithm, making it computationally efficient to draw and save the map of the environment. Sometimes the lines did not appear to be merged when they were supposed to. However, it was later found the problem was associated with the resolution of my laptop screen and when zoomed into the map, the lines were indeed merged and being represented as one.

Figure 1 displays an instance of the mapping procedure where it is trying to map one of the environments provided. Here the grey points represent the points of all the data clouds processed so far and the red line represents the extracted line feature. Here the red lines for all the previous instances have been displayed by simply changing the marker_id parameter every iteration.

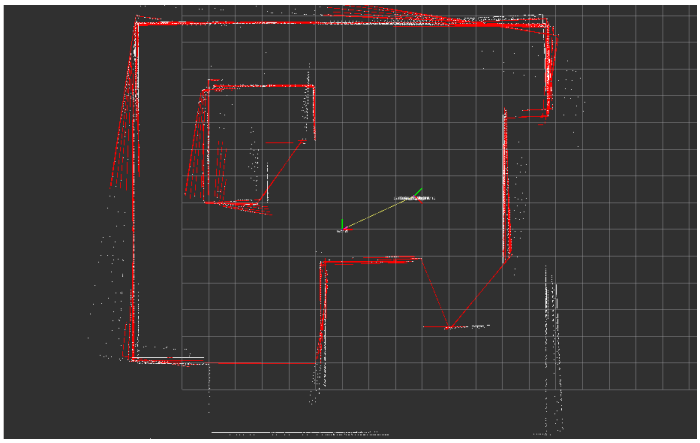

Figure 1: Output of the algorithm while operating (the red lines appear to be separated/not merged, however when you run the program and zoom in, they are the same - problem is with the resolution of my laptop screen)

The algorithm was tested on another environment, the result of which is shown in figure 2. The threshold values were not changed as the result was visually acceptable.

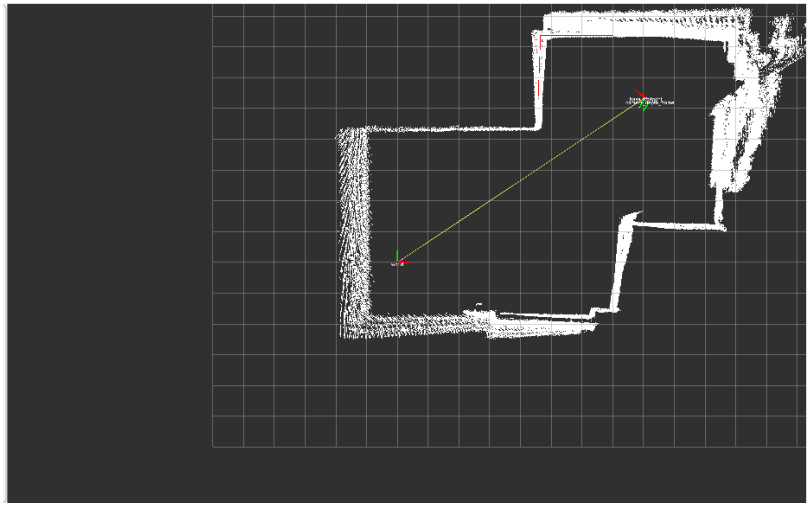

Figure 2: Output of the algorithm for second map (the red lines appear to be separated/not continuous, however when you run the program and zoom in, they are the same - problem is with the resolution of my laptop screen)

Conclusion:-

In this lab-work, a basic line feature extraction algorithm was implemented. The iterative end point scheme was used to implement the split and merge algorithm. The algorithm appears to work however, it should be noted that this algorithm only works in cases in which the points in the cloud are sequentially arranged otherwise the algorithm will yield unacceptable results.