Probabilistic Robotics Lab 2
Split and Merge

Rodrigo Caye Daudt

March 14, 2016

1 Introduction

This report describes a basic implementation of the Split and Merge algorithm. The implementation was written in Python, and is was tested with two sets of recorded data from real robots. These datasets contained information about the robot's odometry system and laser rangefinders. The objective was to use the available data to infer characteristics of the environment in which the robot was moving.

2 Split and Merge Algorithm

The robot's laser rangefinder provides us with with a set of detected points relative to the robot at each cycle. Coupled with the odometry information we are able to calculate the coordinates of these points relative to a fixed world coordinate system (in our case, the (x, y) plane). Although these points are now properly placed in our map, points are not the most appropriate data form for obstacles if we want to perform trajectory planning, SLAM, or other algorithms essential for mobile robotics. Nowadays, many algorithms exist to extract object information from the obtained cloud of points. We implemented the Split and Merge algorithm [1], which tries to split the obtained points into a set of lines, which is a data format much more appropriate for describing a map in mobile robotics.

This algorithm consists on a series of simple steps. First, the algorithm defines a line between the first and the last points in the dataset and calculates the distances between all points to this line. If the farthest point from the line exceeds a given threshold, the line will be split at that point, and the algorithm will be applied recursively to the generated subsets. Once the dataset has been split onto lines according to the defined thresholds, the function checks if the distance between sequential points in every line is smaller than another given threshold, and if so it is split even further.

Finally, the lines with neighbouring points in the dataset are compared regarding distance and angle difference, and if they are similar enough they are merged.

With the right parameters, the Split and Merge algorithm transforms the dataset of points into a much smaller set of straight lone segments, which can be easily analysed for path planning, for example.

3 Implementation

The Python implementation of the Split and Merge algorithm followed very closely the description in Section 2. The *splitandmerge* function had as a necessary input argument the set of points that were to be split into lines. This input was done in the shape of a $2 \times N$ array, where each column represented the (x, y) coordinates of each detected points. The function could also be called overwriting any of the default threshold values used for the Split and Merge algorithm. The output of the function is in the shape of an $N \times 4$ array, where each line contained the (x, y) coordinates of the first and last points of each line segment.

4 Simulation Parameters and Results

We applied the function over two different datasets, called "dataset3" and "dataset2". These datasets had different characteristics (dataset2 was much more dense, for example). Figures 1 and 2 show the result obtained using the parameters in Table 1, which were the parameters found to yield the best results for each dataset.

Dataset	split_thres	inter_thres	min_points	dist_thres	ang_thres
dataset3	0.1	0.3	6	0.12	10°
dataset2	0.01	0.3	6	0.12	10°

Table 1: Empirically found parameters for each dataset

5 Difficulties during development

The main difficulties encountered during the development of this work came primarily from the lack of documentation on some of the used functions. For example, the calling of the *publish_lines* function used to draw the lines had to be modified in order to keep the drawn lines on the screen, and information about how to accomplish this was hard to find.

Figure 1: Resulting lines using dataset3

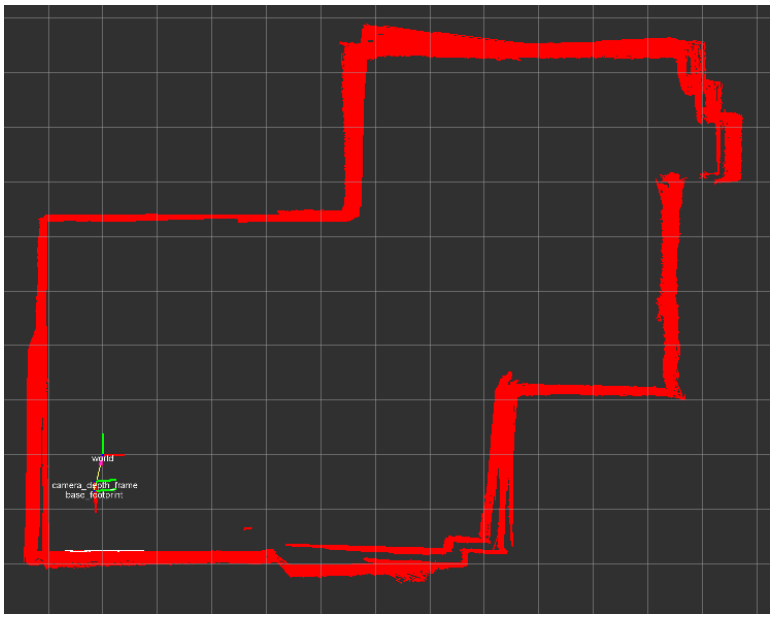

Figure 2: Resulting lines using dataset2

6 Conclusion

The usage of the Split and Merge algorithm was demonstrated, and we observed that it is a simple algorithm able to run in real time applications. We could also observe in the resulting images how errors in the odometry and rangefinder can result in improperly positioned points in space. Finally, we also observed that the results from this method seem to be good enough to be used for other mapping and path planning applications as long as the parameters are well tuned.

References

[1] V. Nguyen, A. Martinelli, N. Tomatis, and R. Siegwart, "A comparison of line extraction algorithms using 2d laser rangefinder for indoor mobile robotics," in *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on.* IEEE, 2005, pp. 1929–1934.