ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Autonomous Systems Lab

Exercise 3
Line fitting and extraction for robot localization

1 Introduction

For a lot of application in robotics, knowledge of the position and orientation of the platform is essential. This exercise could be motivated by an autonomous vehicle hauling goods across the corridors of a warehouse. In order to navigate from one place to another, the vehicle would need to know its position in the warehouse as well as its heading. On its way, it might come across walls, doorways, and racks, all of which would be perceived as measurements located along lines by a laser scanner mounted in a way that its scanning plane is parallel to the ground. Exercise 3 and 4 explore these line features for localization against a known map. While Exercise 3 will show how to extract lines from laser scans using the split-and-merge approach, Exercise 4 will demonstrate how to employ these measurements in combination with a map for robot localization in a Kalman filter framework.

2 Line extraction

A range scan describes a 2D slice of the environment. The range scan returns a set of points with coordinates (x, y). We now want to fit this unordered set of points onto lines.

We choose to express a line in polar parameters (r, α) as defined by the line equation (1) for the Cartesian coordinates (x, y) of the points lying on the line

$$x \cos \alpha + y \sin \alpha = r \,, \tag{1}$$

where $-\pi < \alpha \leq \pi$ is the angle between the x-axis and the shortest connection between the origin and the line. This connection's length is $r \geq 0$ (see Figure 1).

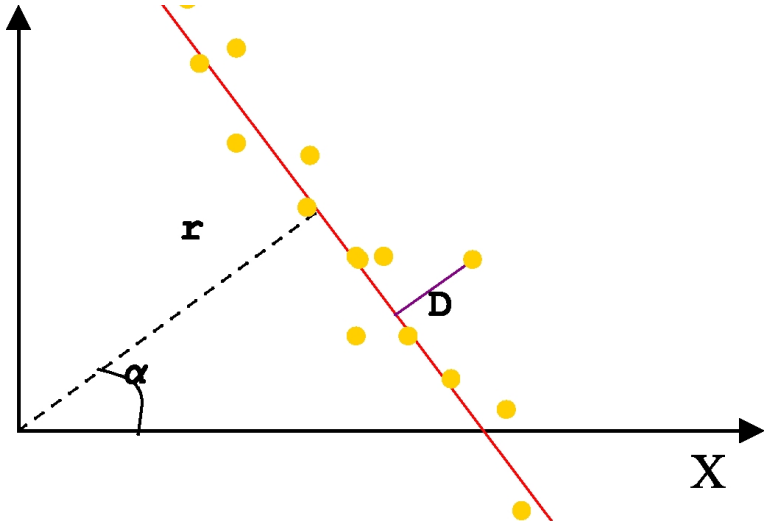

Figure 1: Fitting line parameters: D is the fitting error we aim to minimize expressing a line with polar parameters (r, α)

The error for fitting a set of points onto the line given by (r, α) is:

$$S(r, \alpha) := \sum_i (\underbrace{r - x_i \cos\alpha - y_i \sin\alpha}_{=D((\alpha,r),(x_i,y_i))})^2 , \tag{2}$$

where (x_i, y_i) are the input points in Cartesian coordinates. We find the best fitting for the given set of points by minimizing the errors $S(r, \alpha)$. The solution of (r, α) can be found by imposing $\nabla S = 0$.

Task: Find the solution for r using $\frac{\partial S}{\partial r} = 0$.
If necessary, please find additional information on [1, pp. 244] including a solution for polar input on [1, p. 246].

2.1 Split-and-Merge

We employ the popular "Split-and-Merge"[1, p.249-250] line extraction algorithm to divide the obtained range measurements (points) into segments of points lying roughly on a common line.

Algorithm 1: Split-and-Merge

Data: Set S consisting of all N points, a distance threshold $d > 0$
Result: L, a list of sets of points each resembling a line
$L \leftarrow (S)$, $i \leftarrow 1$;
while $i \leq len(L)$ **do**
  fit a line (r, α) to the set L_i;
  detect the point $P \in L_i$ with the maximum distance D to the line (r, α);
  if $D < d$ **then**
    $i \leftarrow i + 1$
  else
    split L_i at P into S_1 and S_2;
    $L_i \leftarrow S_1$; $L_{i+1} \leftarrow S_2$;
  end
end
Merge collinear sets in L;

2.2 Algorithm in Matlab/Octave

The Split-and-Merge algorithm is implemented inside the function $[\alpha_i, r_i, \ldots] = extractLines(x_i, y_i)$. A crucial part of this function is the line fitting step.

Task: Edit **`fitLine.m`** and follow the instructions to insert the mathematical formula for computing the line regression (line fitting) using a set of points in Cartesian coordinates. Use your solution for r and the following solution for α:

$$\alpha = \frac{\tan^{-1}(\frac{\text{num}}{\text{denom}})}{2} \tag{3}$$

$$\text{num} := -2\sum_i (x_i - x_c)(y_i - y_c) \tag{4}$$

$$\text{denom} := \sum_i (y_i - y_c)^2 - (x_i - x_c)^2 \tag{5}$$

where (x_c, y_c) are the Cartesian coordinates of the (x_i, y_i)'s centroid (see instructions in **`fitLine.m`**).

You then need to implement a splitting of the fitted line at the point where the error is maximal. Edit the function `findSplitPosInD` in file **`extractLines.m`** to find the index where to split the line. You are given the array `d` of all the distances of the points to the line. Either return an index to this array, or, if the line should not be splitted, return `-1`.

As soon as the lines are correctly fitted, the algorithm performs Split-and-Merge and extracts the endpoints of each segment.

Validation: Run **`test/testLineExtraction.m`** to check if the code is correctly completed and how well your fit is. It will show a figure with the measured points and expected lines as together with the found lines and segments in red, green and blue. To test only the `fitLine` function on artifical data use **`test/testLineFitting.m`**.

References

[1] Roland Siegwart, Illah Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots*. MIT Press, 2nd edition, 2011.