LiDART Gap Finding Algorithm Explanation

Team: Xiaoyi Chen, Mustafa Hasekioglu, Nikhil Krishnan, Connor Mckeon

A. Overview

In this lab, we've implemented an algorithm used to find gaps with LiDAR scan data. The code, find_gap.py, is composed of the following components:

1. Read & Process scan data
2. Use DBSCAN to cluster points
3. Using the clustering labels returned in part 2, find the on-cluster central points of each cluster & the boundaries of each in-range obstacle (Figure 1). Notice that we've used a distance threshold (OAT, Obstacle Avoidance Threshold) to filter out the possible noise involved in data points too far away, especially when the car is turning.

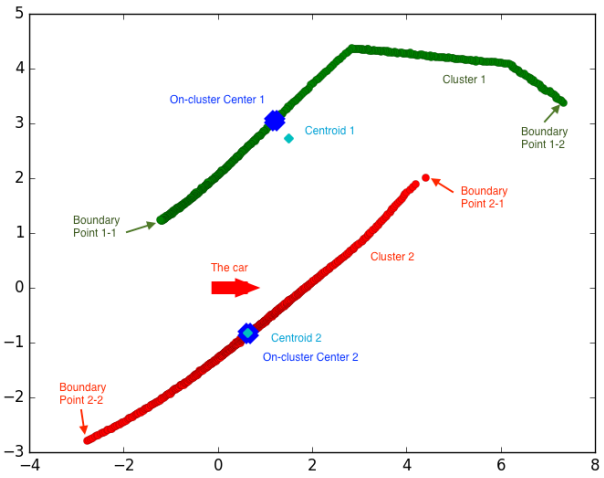

Figure 1. Centroids, On-Cluster Centers, and Boundary Points

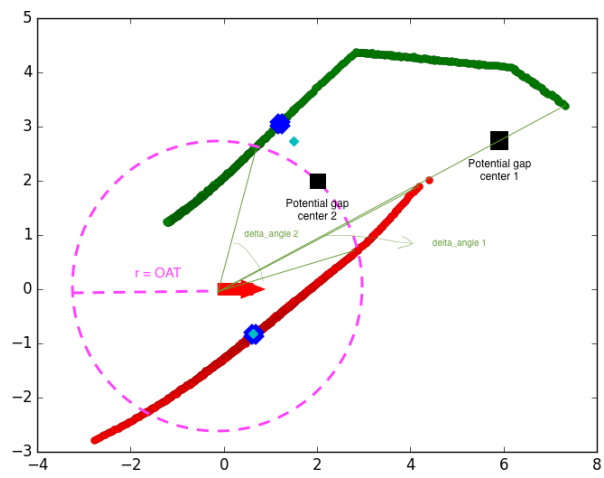

Figure 2. OAT & Delta_angles

4. Find the 2 characterizing points of each gap & save them to a customized message type gap. The widest gap is chosen using the product of the euclidean distance between the 2 characterizing points and the delta_angle of the gap. Notice that we've added another constraint to ensure the gap delta_angle is > 0.5 rad, in order to prevent situations of sharp turns where the gap identified might be on the other side of the turn. If we choose the center of that gap, the car might crash into the inner wall of the track. (Figure 2)
5. Visualize the clustered walls, cluster centers, gap center found as well as the position and direction of the car. The plotting is done through matplotlib.
6. Publish messages about gaps_data on topic '/lidar_gap', gaps_center on topic '/gap_center', and drive_parameters on topic '/drive_parameters'. Notice that the last one has been kept as default, since it will be implemented in a future lab.

B. Pseudo-Code

Read angle & ranges data from received message.
Filter out-of-range range data, and concat angles to their corresponding ranges in a matrix.
Use DBSCAN to cluster the data.
For each cluster returned by DBSCAN:
    Find the centroid using an average of all cluster points.
    Find the on-cluster central point by looking for the closest cluster point to the centroid
    If there are >= 5 cluster points having range values smaller than OAT:
        Ignore this cluster and move on (because it's too small and considered a noise)

Store the obstacle boundaries
Sort the boundaries of obstacles in increasing angle.
From the smallest angle (angle_min) to the largest angle (angle_max):
Append angle and distance of detection boundaries and obstacle boundaries to a gaps_data message;
Append corresponding x and y coordinates to this message as well.
Find the delta_angle and euclidean length of each gap, and save it to gaps_data.
Find the widest gap with at least 0.5 radian delta_angle. Select that as the widest gap.
Publish messages of gaps_data, gap_center and drive_parameters.

C. Results

Using Gazebo simulator (Figure 3) and matplotlib (Figure 4), we can see that at this turn shown in Figure 1, we've managed to:

1. Identify the 2 walls each as a cluster. (plotted in green and red)
2. Marked the on-cluster center of it. (blue dots)
3. Calculated the center of the widest gap, which will be used for path planning. (black dot)

*Notice that in Figure 3, the car is represented as the red arrow pointing towards the right.

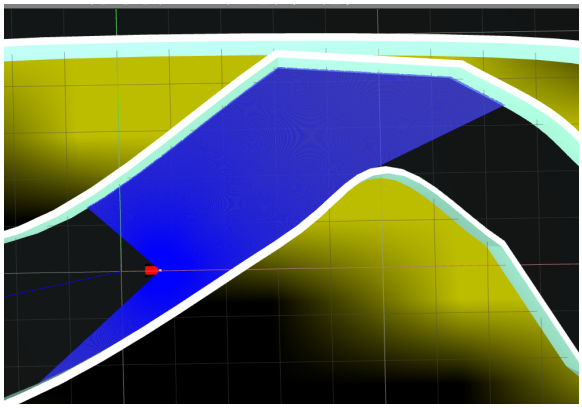

Figure 3. Gazebo simulator

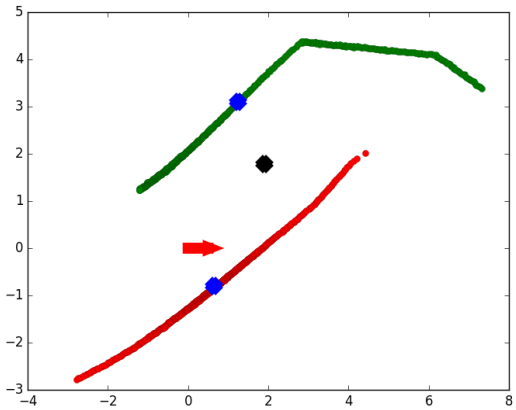

Figure 4. Matplotlib plot

We also showed the LiDAR data and identified gap center in RViz, as can be shown in Figure 5, where the yellow sphere shows the gap center, and the red horizontal line shows the moving trajectory of the car. Since we haven't implemented the path planning algorithm, the car moves straight to the right.

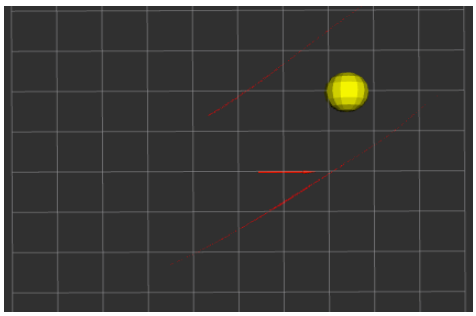

Figure 5. Rviz Visualization

D. Reference

Ayoade, Adewole, et al. "Laser-Based Gap Finding Approach to Mobile Robot Navigation." *2016 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, 2016, doi:10.1109/aim.2016.7576876.