Robotics Practical 1: Assignment 3

Laura M Quirós (s4776380)
Adriana A. Haralambieva (s4286103)

March 10, 2024

Q 1. For the report: describe how you determined your cluster threshold value and write down its value. As with any academic reporting, a fellow student should be able to reproduce your process from your written description.

To determine the different clusters, we go through all the points from the data with a for loop and check whether that point is closer to the previous one than the defined threshold. If it is, we add it to that cluster, and if not, we create a new cluster. We set the cluster threshold to 0.1. We decided to leave this value after testing a few different scenarios.

The most telling scenario was putting two objects in different lanes, but almost on the line separating them (so, very close to each other but on different lanes). This is because we wanted to be able to differentiate between the lanes being occupied in a good enough manner. 0.1 as a threshold seemed to do this quite well. This can be seen in the purple and orange clusters in figure 2.

There is also another scenario we consider, which is if there is an object behind the robot that crosses the x-axis. Because of the way in which the LIDAR points are collected, if unaccounted for, this would lead to two separate clusters. In order to fix this, first we check if there are two or more clusters. If yes, we would check whether the first and last one are the same cluster (this scenario is shown in Figure 2, the blue cluster). If the first point of the first cluster and last point of the last cluster are closer than the threshold, we would assume that it is the same cluster. We pop a cluster from the cluster list and add its points to the first cluster.

Q 2. Include in your description a screenshot of a situation in Gazebo you tested and the corresponding cluster visualization from *plot_clusters.py*.

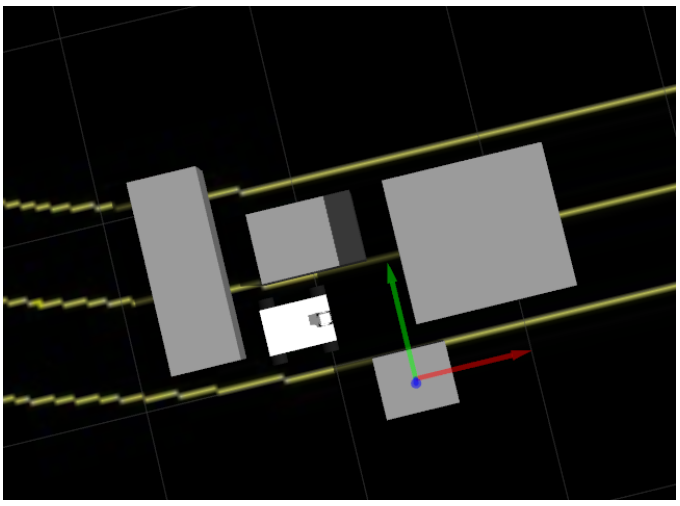

Figure 1: The tested situation from Gazebo.

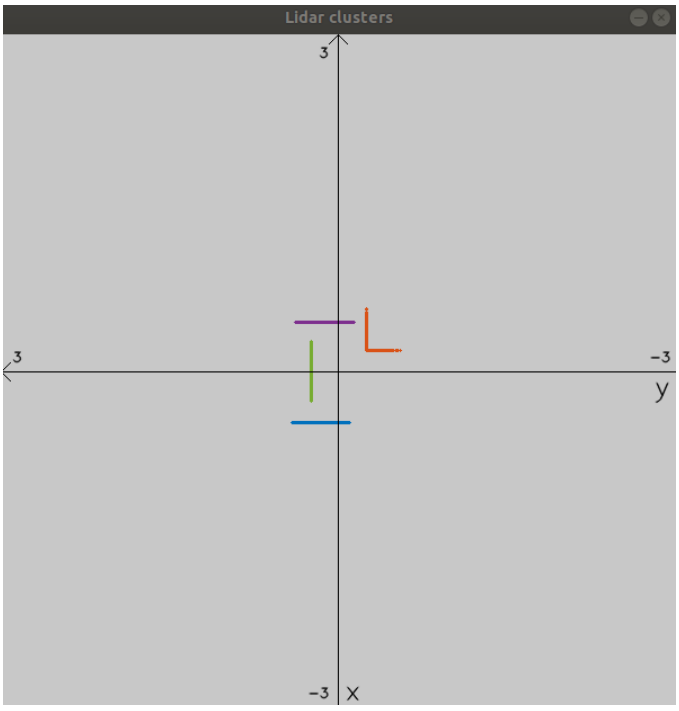

Figure 2: Cluster visualization of the tested situation.

Q 3. For the report: include a description on how you implement the speed adjustment for case 2. Which detections signify to start and end the speed adjustment respectively? How do you maintain a similar speed?

In order for the speed adjustment to be triggered, we check in our `laser-callback` function whether there is an object (which we tested to be another robot) in front of our robot closer than 2 meters. We do this by using a new function `is-front-robot-close`, in the object tracker class. This function is given all the points, which then filters. Out of the points between the -0.25 and 0.25 which define the center lane, if any of them has an y value under 2.0, we consider the robot in front to be too close.

If the function returns true, we first check left and right lane separately - whether they exist in the first place in this situation and whether they are free. We check if they exist by looking at the lane follower class. In the fields of the lane follower we store all the final lines delimiting the lanes under the field name "`lines`", which is a dictionary. To check if the lanes are free, we look at the `lane_occupied` dictionary of the object tracker class, which tells us if the lane is free in the next 2.5 meters.

For case 2, we assume that none of the lanes are free/exist, so the robot cannot move out of its current lane. In that case we want to decrease the speed: We first check whether the current speed is higher than 0. If yes, the robot decreases its current speed with 2.5%. This seemed to be slow enough to not cause an abrupt stop. After this, every time we check whether the robot in front is still too close. If it is not anymore, we increase the speed with 2.5%. The increase happens every time that `is-front-robot-close` is not true and speed is under 0.7, which guarantees that once we have switched lanes, we'll go back to the initial speed. We clip the speed value to be maximum 0.7 (this is the initial speed that we need to maintain according to the assignment) and minimum 0. In this way by slightly increasing and decreasing the speed, the robot manages to follow the robot in front a have a safe distance.

Once a free lane is available, the robot would immediately switch to it, and gradually increase its speed to 0.7 (if it was less than this). Before switching we ensure that the angles of the lines are small enough so that the switching can be done smoothly.

This behavior is also triggered if we request a turn and we can't make it at the moment. We do so by having `request_turn_left` and `request_turn_right` boolean variables which are included in the if statements to trigger speed adjustment and lane-switching.