LiDAR lab

Niraj Basnet
basnetn@oregonstate.edu

March 30, 2020

The objective of this lab is to get you to work with LiDAR data and the simulator, and create new messages, by implementing a useful and simple perception algorithm: gap finding. This document covers two assignments: gap finding in the simulator and gap finding on the car.

1 Setting up the simulator

Start from a clean workspace, which you've setup using the instructions from the ROS tutorials. In a separate folder, git clone the skeletons repo.
git clone https://github.com/sabotagelab/f110-skeletons-spring2020.git
If you check its contents, it should have a `systems/` folder and a `simulator/` folder.
Now continue with the instructions in the Readme file of the repo, starting with the commands to copy the contents of the repo into your workspace. Basically copy every packages in the downloadred repo to your workspace and make sure to build the workspace using catkin_make.

Note:There are two simulators available in the skeletons repo inside the simulator folder. One is racecar_gazebo_simulator and other is racecar_simulator. Gazebo is the standard simulator that comes with ROS. However, it is not that reliable and consistent and sometimes you might have to relaunch it multiple times when it crashes. So, we will be using a lightweight 2D simulator inside the package racecar_simulator which uses RViz as display interface and kinematic vehicle model to simulate vehicle dynamics. Hence, it has a pretty good refresh rate and launches quickly without any crashes. Therefore, this is set as the standard simulator for this class.
However, it doesnot mean that you cannot try Gazebo. The package is included in the repo

and you can try to run the car and experiment few stuffs on your own by following the reference manual. *But remember it won't be used for grading your labs or assignments.*
Follow the instructions in the Readme file inside the racecar_simulator package to launch the simulator. Familiarize yourselves with the simulator and its features; do all that you can from the Readme file as well as Reference Manual. Try to load different maps into the simulator and operate your car in keyboard control mode or navigation mode.
Bonus 1 points on the final grade Dig into the simulator code to find the dynamical model of the car: this is the code that accepts steering and acceleration commands, and moves the car accordingly. Copy that and turn it into a stand-alone piece of code. Submit that as a separate folder in your workspace, called `studentname_dynmodel`

2 Finding gaps in simulated laser data

When running the simulator, one of the ROS nodes publishes LaserScan messages on topic /scan.

- Which node publishes these messages?

In this lab you will create a *gap finding* algorithm. *Gap finding* refers to the task of finding a gap in one LiDAR scan. This can be used in *gap navigation*. Gap navigation, *in this course*, refers to the path planning algorithm which repeatedly finds a 'good' gap, and sets the next waypoint to be the middle point of this gap.
What type of gap is best for these purposes? One option is the *widest* gap: a wide gap allows for more error in localization and actuation, and potentially more room for lateral maneuver should a last-minute modification be necessary. On the other hand, the *deepest* gap is the gap past which there is the most space. A deep gap allows for more time to plan the next move, and potentially, aiming for the deepest gap can result in faster overall driving since the car has more time to break. On the other hand, the deepest gap might be too narrow for the car's localization abilities. A balance is usually necessary between these choices.
In this lab, you will implement a gap-finding algorithm and test it in simulation. This includes two steps: finding a good gap in the laser scan; selecting the waypoint in the gap

1. Your solution should be in a new package `studentname_gap_finding`. The node that implements the solution should be called find_gap.

2. You may assume that the car is driving down a closed track, and that the track is obstacle-free (other than the track's walls, of course). Take a look at `./simulator/racecar_simulator/maps/map_trackporto.pgm`. If you want to look a 3d view at Gazebo,then use the world file from the path `./simulator/racecar-simulator/racecar_gazebo/worlds/track_porto.world` for an example of such a track. This is the actual track that the 3^d F1/10 competition took place in, by the way!

3. You can either drive the car manually in keyboard or joystick mode on the simulator or use some basic navigation algorithm to make it go around the track. This way you can observe different sorts of laserscan data which you can use for finding gaps.

4. *K-means clustering* might be one way of detecting gaps in one LiDAR scan. There are many implementations of k-means online, including in Python. E.g., checkout the scipy library's version at `https://docs.scipy.org/doc/scipy/reference/cluster.vq.html`. This is a suggestion. You may want to use a different code for k-means, a different clustering algorithm(e.g,DBSCAN), or no clustering at all! Whatever you choose, experiment with it. Most importantly, you should understand its failures, and its important parameters.

5. If you do use some form of clustering, think about how many gaps you expect to find under the above track constraint, and what this implies about the expected number of clusters (k) in a scan. Should use set k equal to the expected number of gaps? Or should you use more clusters than expected number of gaps, then select the 'best' one?

6. If you use clustering: track walls might be identified as a cluster: the feature vectors from consecutive wall points are close in Euclidian distance. The angle should help you eliminate these.

7. Your node should *publish* all the gaps it finds. The topic should be called `lidar_gaps`. The message type is a new type that you create and is called `gaps` (review the ROS tutorials for how to create message types). The message type `gaps` will be defined inside the `teamname_gap_finding package`. Some things to think about:
   - The number of gaps will vary from one scan to the next, unless you impose/expect there to be the same number of gaps.
   - The width of a gap will vary gap to gap, and scan to scan. So the data struct used to implement message `gaps` must allow this.

8. Your node will also publish, on topic `gap_center`, the centerpoint of the best gap. The message type is a simple `geometry_msgs::Vector3`, which is a built-in ROS data type.

9. A priori, you can find a gap in the i^{th} scan using only i^{th} scan data. You can improve over this by also taking into account the data from the last ℓ scans, where ℓ is a tunable parameter, say $\ell = 2$ or 3. The idea is that the centers of two consecutive gaps will not be too distant from each other, so this might limit the search space for the next gap. This introduces a *smoothing* effect on the planned trajectory.

10. **Bonus 1 points on the final grade**: Based on the gaps you find, implement a simple navigation algorithm to follow the gap-center and travel around the track autonomously. You can select any map of the track from the simulator package. Name that file as studentname_gap_follow.py or student_gap_follow.cpp and submit that along with rest of the code.

Evaluation:
We will run

```
$ cp -r studentname_gap_finding our-work-space/algorithms/
$ catkin_make
$ roslaunch gap_finding gap_finding_sim.launch
$ rosrun studentname_gap_finding find_gap
```

In RViz, we will display the gap center, obtained from messages published on `/gap_center` topic, to see how well you are finding the gaps. To do the visualization, we will use the python script `visualize_gap_finding.py`, which is also in the lab skeleton. Feel free to use it yourself. If you do, you may need to modify the topic to which it subscribes in order to get the gap center messages.
We will examine the messages published on topic `lidar_gaps`, and the structure of the message `studentname_gap_finding::gaps`
You will also submit a 2-pages single-space explanation of the gap finding algorithm, including figures. Call this document teamname-gap-finding.pdf. Please keep the explanation simple and to the point. Follow this structure:

- First, pseudo-code of the algorithm (similar to the presentation you saw in your introductory Algorithms/Data Structures class). Don't shy from mathematical notation!
- Explain the algorithm in one paragraph, referring to the lines of the pseudo-code.
- Highlight the important user-set parameters of the algorithm, and explain their effects on performance (e.g., if we increase the range cut-off, the car will be looking further ahead, but might not locate itself best to go through the center of the closest gap.)
- A figure to help explain the pseudo-code, showing the important parameters.

3 Gap-following in a race

The winning team in the F1/10 Grand Prix in Torino, Italy, in October 2018 used gap-finding as their navigation algorithm. Read about their experiences here: `https://medium.com/asap-report/experiences-and-thoughts-from-the-3rd-f1-10th-competition-2c46508e2719`