Lab 2. AEB and PID control

Due date: Friday Mar. 27, 2020 on Course Site
Project Lead: submit codes and peer evaluation form
Subscriber: submit lab report and peer evaluation form

Major Tasks:

0. Review and reinforce ROS skills;
1. Design automatic emergency brake (AEB) program in simulator;
2. Learn PID control and design wall following algorithm in simulator.

Requirements: Make sure your codes include the author names, course and lab numbers, date and other related info in the comments. Your codes shall follow the Python-ROS coding style guide: http://wiki.ros.org/PyStyleGuide

Preparation before the lab:

- Get familiar with Linux command lines, please go through the tutorial here: https://ubuntu.com/tutorials/command-line-for-beginners; Also check out a free book on ROS tutorial: https://www.cse.sc.edu/~jokane/agitr/agitr-letter.pdf Thanks to Velin Tarkochev for recommending it.
- Go through the tutorial: Using rqt_console and roslaunch: http://wiki.ros.org/ROS/Tutorials/UsingRqtconsoleRoslaunch
- Watch the three UPenn tutorial video tutorials in Session 2 at http://f1tenth.org/code.html.
  - Lab exercise: Driving Straight – distance finder and sensor messages
  - 2.2 Localization – coordinate system and scan matching
  - 2.3 PID Control – Proportional, integral, and derivative control

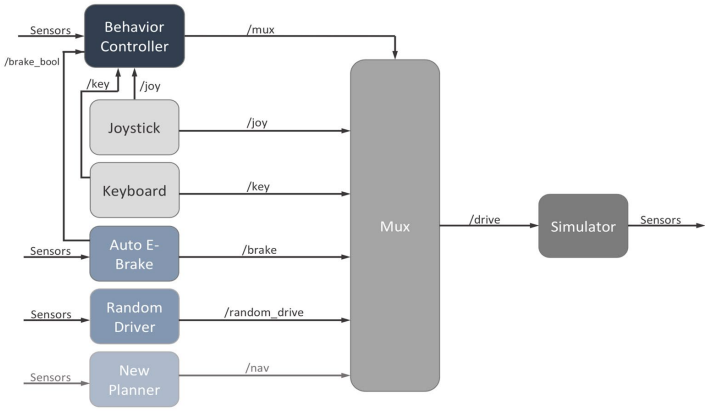
We are not going to deal with the scan matching in this lab, but try to understand the concept and pay special attention to messages and coordinate systems of the Lidar sensor and Odometer sensor (in VESC) . We will be using PID control in Part 1-2 of Lab 2.

Part 0. Learn debugging tools in ROS:

0.1. Create a new ROS workspace called f110_team#. Git clone the UPenn f110_simulator here: https://github.com/f1tenth/f110_ros.git

0.2. Go to the f110_simulator folder and follow the README.md in that folder to catkin_make, source and run the f110_simulator: $roslaunch f110_simulator simulator.launch

0.3. This shall launch a similar simulation as racecar_simulator in Lab 1. Run the debugging tools rqt_graph, rqt_console, and rqt_logger_level with the simulator, and try to understand how the simulator works. Refer to the following figure included in the simulator package too:



0.4. Create a ROS package under /f110_ros and call the package "roslab2team#" (replace # by your team number throughout the lab). Copy your subscriber+publisher code you had written in Lab 1 to roslab2team#/scripts subfolder, catkin_make, source, and run the new node while running f110_simulator. Move the car in Rviz. Output the rqt graph and the rqt_console message.

Your report shall include your experience working with the rqt tools and answer the following questions:

Q0.1. What are the rqt_console and rqt_logger_level tool? What are the different levels of message you can select in rqt_logger_level? How can they help with understanding or developing the simulator?

Q0.2. What are the rqt graphs with and without running the new node in roslab2team#?

Q0.3. What are the aspects that you think the f110_simulator can be improved in comparison to turtlesim?

Q0.4 (Optional bonus 5 points) Try rqt_plot with the simulator by using the random mode to move the car in the open space and plotting the position (x,y) and velocities (linear.x, angular.z). Explain your procedures and show your results saved from the plot.

Your code submission shall include: the csv file of the rqt_console in Step 0.4.

Part 1. Automatic Emergency Brake:

1.1 Review python tutorial http://cs231n.github.io/python-numpy-tutorial/ and the tutorial on Classes and Objects here: https://www.learnpython.org/en/Classes_and_Objects. A more detailed documentation/guide on python classes is here: https://docs.python.org/3/tutorial/classes.html

1.2 Under the f110_ros/safety subfolder, find the python code template in /scripts. Your task is to modify the template file and implement a new node "team#safety_node" for automatic emergency brake (AEB). Refer to the pdf file in the f110_ros/safety subfolder for technical details. Also see the python file in /waypoint_logger/ for examples of subscribing to Odometry and LaserScan messages. Pay attention to the different messages needed and take care of division by zero as well as bad ranges data. Make sure you modify the comments to reflect your changes and add necessary comments to explain your algorithm. Test the new node with the f110_simulator using the random walk in the middle of the building on the map. Demonstrate your work to the TAs when you complete this task.

Your report shall demonstrate/state that both team members work through the Python tutorials. Make sure to also answer the following questions.

Q1.1. What are the variables and functions defined in your Safety class?

Q1.2. What are the difference between the two layers of twist and pose in Odometry message in topic "/odom"? What are the fields in the lower twist and pose of Odometry?

Q1.3. How do you find the angle of each data entry in "ranges" of LaserScan in topic "/scan"?

Q1.4. What is your experience of running the simulator with the AEB? How does the AEB behave when you drive backwards? What difficulties did you encounter in Part 1 of Lab 2?

Q1.5. In real-world AEB systems, what are the two types of error in false detection (wrong classification)? What are the impact of the two types of errors?

Your code submission for Part 1 of Lab 2 shall include: the python code safety_team#.py.

Part 2. Wall Following:

2.1 Review the principle of PID control. Many PID tutorials are also available online. One good page with Python code example is here: https://onion.io/2bt-pid-control-python/.

2.2 Under f110_ros/wall_follow subfolder, create a new folder "scripts" and copy the template python code in the "src" folder to /scripts. Change your file name to team#WallFollow.py. Follow the instructions in the Wall_Following_Lab.pdf file and modify the code to implement a new node team#WallFollow_node. If your team number is odd, follow the left wall. If your team number is even, follow the right wall. Test your node on the f110_simulator and demonstrate your work to the TAs via zoom.

2.3 If you are interested in tuning your car with VESC tool or testing your WallFollow node on the car, please let the instructor know. We will make special arrangement to work on the car depending on our access to the Lab room in the next week.

Your report for Part 2 of Lab 2 shall include answers to the following questions:

Q2.1 How does PID control work? What are the differences among the three types of control: proportional, integral, and derivative? How do you select the parameters of each control method?

Q2.2 How do you fine tune the VESC on your car? After watching the VESC tuning video, what do you learn about the motor and the VESC?

Q2.3 Explain your wall following algorithm and its relationship with the PID control. How did you test your WallFollow node in the simulator? How did the parameters affect the results?

Q2.4 How do you test your WallFollow node and AEB node simultaneously in the simulator? What is your observation?

Q2.5 (Optional for 10 point bonus) Derive the Laplace transfer function for the PID controller with a servo motor. The plant consists of the motor and an inertial load, and is modeled by a second order differential equation. Comment on how the parameters can cause the system to be unstable.

Code submission for Part 2 of Lab 2: zip or tar all files under the folder /f110_ros and submit the zip/tar file. Note that your AEB node shall also be included. We will test both nodes in your submission.