Lab 4. Pure Pursuit Tracking Algorithm

Due date: Friday May 1, 2020 on Course Site
Project Lead: submit codes and peer evaluation form
Subscriber: submit lab report and peer evaluation form

Major Tasks:

1. Learn SLAM principle and algorithms for robotics;
2. Utilize Particle-Filter SLAM in the f110 simulator;
3. Design and test pure pursuit tracking algorithm on race tracks.

Requirements:

1. Make sure your code include the author names, course and lab numbers, date and other related information in the comments. If you use code or code segments found from the internet, then clearly refer to the source of the code. By having your names in the comments, you agree that you have put effort in developing the code. Your codes shall follow the Python-ROS coding style guide: http://wiki.ros.org/PyStyleGuide
2. Do not change the parameters in the simulator as those are designed to match the real f110 car. However, you may change the structure provided in the template .py file to fit your algorithm.

Your report shall include answers to the following questions:

Q1. What is SLAM and what are the three major approaches to solving the SLAM problem? Give a brief review of the three approaches and typical works.
Q2. What is the open SLAM project? What are the resources in the repository that you may leverage?
Q3. What is a quaternion? What are the Euler's angles? How are the pose and laser scans transformed between the base_link (device) frame and the map frame?
Q4. What is the Google Cartographer? How is cartographer implemented in ROS?
Q5. What are the details of your pure pursuit algorithm? If you tried different parameters and control or racing line strategies, provide comparisons between the different settings.
Q6. How does your pure pursuit algorithm perform in comparison to the gap following method?
Q7. What are the four columns of data in the waypoint logger file? What is the reference frame of the data? What does it mean if the entries on column 4 are zeros? How did you edit your waypoints?
Q8. What difficulties and problems did you encounter in Lab 4? What did you try to solve them? What did you learn from this experience?

Code submission shall include: the team#PurePursuit.py file, map files (.pgm and .yaml) if they are not in the packages, waypoint logger .cvs file, and any (optional) waypoint display or modification codes or any video that you make for performance comparison.

Step 1. Create a waypoint logger for a map:

1. Git clone the particle_filter package from https://github.com/f1tenth/particle_filter.git under the subfolder /src of your catkin workspace; Follow the readme.md in the package to install all dependencies and the RangeLibc library – note the ROS version has to be changed to melodic. The summary of the commands are listed here:
   i) Your-catkin_ws/src$ git clone https://github.com/f1tenth/particle_filter
   ii) $cd ..
   iii) $sudo apt-get update

iv) $rosdep install -r --from-paths src --ignore-src --rosdistro melodic –y
v) $sudo apt install python-pip #if you don't have pip yet
vi) $sudo pip install cython
vii) Under src folder: $git clone http://github.com/kctess5/range_libc
viii) $cd range_libc/pywrapper #note the README.md has an extra 's' (error) here
ix) ./compile.sh #run this on Ubuntu CPU/laptop
x) (optional) If on Jetson GPU (the car), run this instead: ./compile_with_cuda.sh
xi) (optional) If you would like to have the trace on option, run $sudo TRACE=ON python setup.py install

2. Select a map in the simulator package or the particle_filter package and add the map in both packages' launch files by:
   i) In the f110_ros package: change the map name in simulator.launch under f110_simulator as <arg name="map" default="$(find f110_simulator)/maps/gap1.yaml"/> or <arg name="map" default="$(find particle_filter)/maps/xxxx.yaml"/>
   ii) In the particle_filter package: in map_server.launch, change the map file name to be the same as that used in f110_simulator/launch/simulator.launch.
   iii) Also in particle_filter package localize.launch: comment/uncomment these lines:

	Odometry setting (line 33-34)	Range_method setting (line 47-48)
On Ubuntu CPU	<arg name="odometry_topic" default="/odom"/>	<param name="range_method" value="pcddt"/>
On Jetson GPU	<arg name="odometry_topic" default="vesc/odom"/>	<param name="range_method" value="rmgpu"/>

   Check the max sensor range in meters and make sure it matches your Lidar model.

3. Create a path in the Home folder called "~/rcws/logs/" (replace # by your robot number) or a path name of your choice. In different terminals, source the devel/setup.bash environment and run:
   a) $roslaunch f110_simulator simulator.launch
   b) $roslaunch particle_filter localize.launch
   c) $rosrun reactive_methods reactive_gap_follow.py (or wall_follow wall_follow.py, press button 'n' in the simulator terminal to run the car. You may also use keyboard to run the car).
   d) $rosrun waypoint_logger waypoint_logger.py

   The waypoint_logger node will begin logging waypoints into a .csv file and save the file in the path "~/rcws/logs/". If you do not have this path, error will occur. If you would like to change the path, then modify the default path and file name in waypoint_logger.py.
   Note: If you get an error message like "cannot locate node of type particle_filter" or the like, then locate the file and check the file permission. Run chmod +x if needed.
4. Let the car run for a little over one complete circle and stop the waypoint_logger node. No need to log waypoints for multiple circles. Otherwise, it becomes a burden to edit the file and it increases the complexity for pure_pursuit.py to search through all waypoints.
5. Edit the waypoint logger file (.csv) to generate the best racing line for the track. To visualize the list of waypoints you have, and to visualize the current waypoint you are picking, you need to publish the *visualization_msgs* messages and enable (add) them in RViz. The provided waypoint_display.py gives a rough example of how the waypoints are displayed in RViz. Feel free to modify/improve it. More information about Marker in RViz is available here: https://www.programcreek.com/python/example/88812/visualization_msgs.msg.Marker and http://wiki.ros.org/rviz/DisplayTypes/Marker.

6. You are free to use other methods to edit the waypoints. Please describe your method if you choose other methods.

Step 2. Design and test pure pursuit algorithm:

1. After you designed the waypoints in the waypoint.cvs, use it in the pure_pursuit.py file on the line *filepath = home+'rcws/logs/xxxx.csv'* .
2. Design your pure pursuit node using the template in the f110_ros package under /pure_puresuit folder, then in separate terminals, run
   a) $roslaunch f110_simulator simulator.launch, press button "N" in command window when launched;
   b) $rosrun pure_pursuit pure_pursuit.py
   c) Run timer.py provided in RACE 1 if you would like to measure the race time.

For racing line strategies and pure pursuit algorithms, see references:

1. MotorTrend Channel: the Racing Line Ep. 4 https://www.youtube.com/watch?v=3HwnJ1VkWA4
2. R. Craig Coulter, "Implementation of the Pure Pursuit Path tracking Algorithm," CMU Tech report, 1992. https://www.ri.cmu.edu/pub_files/pub3/coulter_r_craig_1992_1/coulter_r_craig_1992_1.pdf
3. J. M. Snider, "Automatic Steering Methods for Autonomous Automobile Path Tracking," CMU-RI-TR-09-08, 2009 https://www.ri.cmu.edu/pub_files/2009/2/Automatic_Steering_Methods_for_Autonomous_Automobile_Path_Tracking.pdf