ECE 350/450 Intro to Robotics, Lab 6

**PFSLAM / Pure Pursuit**

Nicholas LaRosa and Nolan Sornson

Department of Electrical & Computer Engineering

Lehigh University

{nil318, nls221}@lehigh.edu

Date: 4/16/2021

**Abstract**

In this lab, we started working with the particle\_filter project to create waypoints on a track using odometry and teleop, and we created our first deliberative navigation algorithm: a pure pursuit algorithm working with a set of waypoints and the odometry data to position the car independently of the LIDAR sensor data. Once this algorithm was working correctly, we then installed Google Cartographer on the car to create a map of our track, and we used our created map with a MATLAB script to create waypoints to run on the car with our pure pursuit algorithm. This report will detail our procedures and results, as well as some issues we encountered in the process.

**Introduction**

* Particle Filter - an algorithm that takes data coming from the LIDAR and interprets it in the form of particles, and it attempts to gain information from the particle field by making inferences and filtering the data through the particles, in our case, to create a map.
* SLAM - Acronym for Simultaneous Localization and Mapping, an algorithm that uses mapping (in the form of a particle filter or similar) and combines that with localization techniques to determine the position of the car in relation to its surroundings. It is a robust algorithm for autonomous vehicle navigation.
* Pure Pursuit - A navigation algorithm that takes a map and a set of waypoints and tries to track them using data from odometry. This is a deliberative algorithm since it takes preemptive information about its surroundings and uses this information to make decisions on its navigation actions. The only feedback involved in the algorithm is that which comes from odometry, so the algorithm does not (at least in its simplest form) make navigation decisions on-the-fly based on any type of obstacle/LIDAR data.
* Waypoint - A position goal that is fed to the pure pursuit algorithm to make decisions on steering angle in order to reach the goal and advance to the next one.

**Procedures**

1. Part 1
   1. For part 1, we followed the given instructions exactly. We chose to use the gap2 map to test our initial algorithm on.
   2. We ran waypoint\_logger using our gap follow algorithm to generate an initial set of waypoints. However, we didn’t actually have a pure pursuit algorithm written when we collected this data.
   3. Initially we struggled with how to approach pure pursuit, so the next step we took was to investigate ginput() in MATLAB to create a short script to easily design waypoints. Our script uses the resolution and origin information from the auto generated map.yaml, reads in our image and lets the user click on the points using ginput() until they press enter. Once enter is pressed, the points are converted from pixel coordinates to car frame coordinates and a csv file is generated. We also are writing a third data value for the target speed to be used to that waypoint. This was one idea we had to control the speed, but is largely unused at this point in time.
   4. Our epiphany came when we realized we could use the rotation equations in [1] to rotate our points into the device frame of view. The key detail is that, because we have to constantly assume we are in the car’s frame, we need to rotate opposite of the direction we are turning.
2. Part 2
   1. To install Google Cartographer we followed the instructions exactly. We did not write a shell script.
   2. We opted to use VNC/Remmina to start and use Cartographer. However, the procedures outlined in the lab for creating a startup application did not work. Instead we wrote a simple startup shell script that runs the command and saved it to the home directory. Starting vnc is as simple as starting the robot with a monitor and running that single script. This script may be able to be started using ssh, but we did not test that.
   3. To begin mapping, we discovered that we need to mark our origin to be able to repeat our starting position exactly. Masking tape and alignment using the tiles on the floor was sufficient.
   4. We discovered that driving too quickly while mapping resulted in poor quality maps. Joy\_teleop.yaml was adjusted to limit the car’s top speed to 1m/s. We attempted to drive as smoothly as possible while mapping.
   5. We mapped both the oval test track and the race 2 track. We found that 2 laps around the track was usually sufficient to get a good map. Our reasoning for this conclusion is that we ran 2 laps around the race 2 track, saved a map, and then continued another 2 laps. Comparing the two maps we observed no consequential differences.

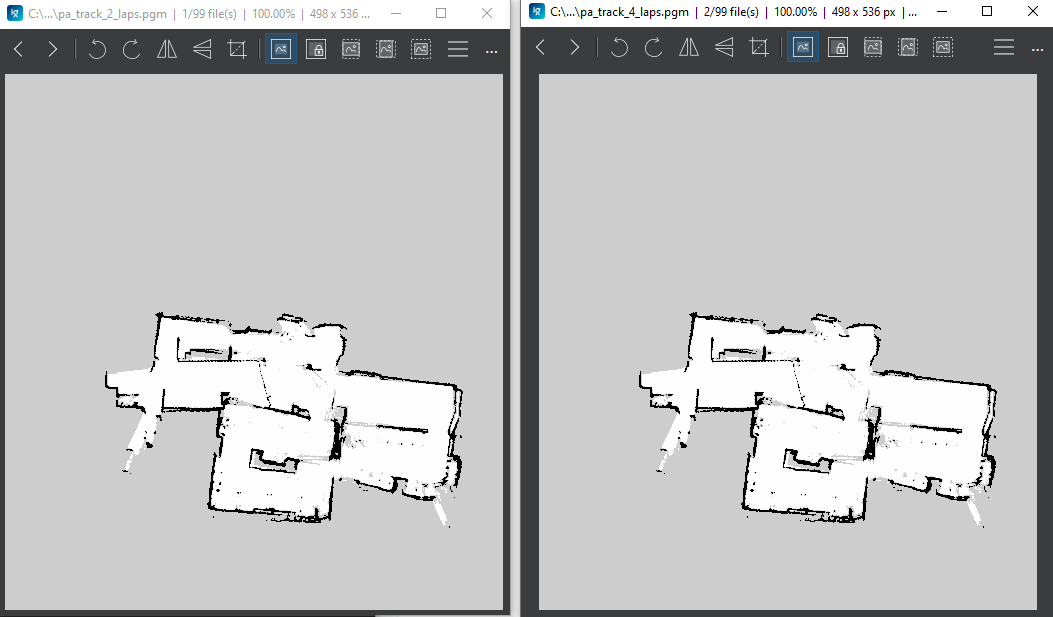
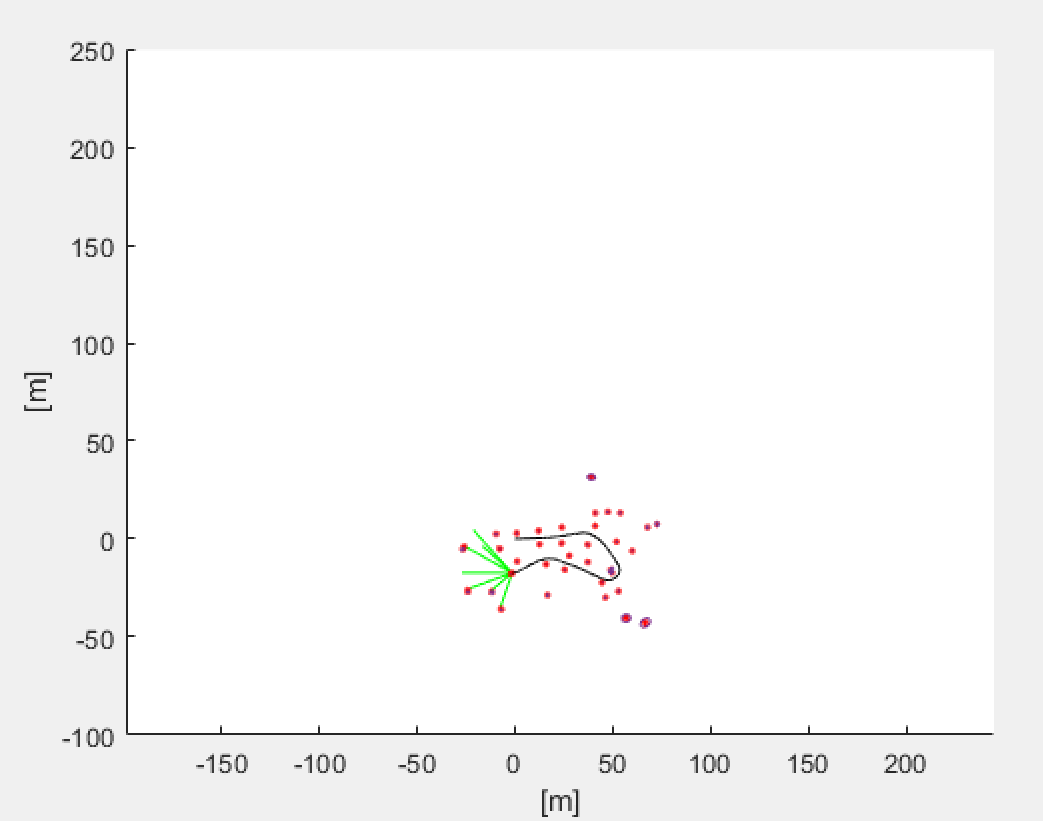


Figure 1: 2 Laps vs 4 Laps

* 1. While mapping, we observed that Cartographer could not keep up. After completing two laps, we found it best to leave the car stationary with Cartographer still running. When this is done, the SLAM algorithm will continue to analyze the collected data and continue to align the submaps. After two laps, we found the car needed between 5-10 minutes stationary to fully complete its alignment.

**Analysis and Results**

1. Part 1
   1. Q1.1. 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.
      1. EKF-SLAM
         1. EKF-SLAM attempts to predict the robot’s next location and when it reaches that point it takes an observation of its current position/state and adjusts its trajectory accordingly.
      2. Particle Filter SLAM
         1. Uses data collected over multiple laps to identify recurring landmarks and filter out erroneous data. Correctly identified landmarks are weighted as more important than other range scans and during resampling, smaller weighted particles are dropped. This helps generate a map with low noise.
      3. FastSLAM
         1. Is similar to particle filter scan as it tries to collect data and align submaps. However, instead of moving individual particles, fastSLAM attempts to rotate the axis of the individual submaps to converge submaps.
   2. Q1.2. What is the open SLAM project? What are the resources in the repository that you may leverage? For ECE450 students, pick the FastSLAM package in openslam-org.github.io (pkg of Tim Bailey) or https://pythonrobotics.readthedocs.io/ and try it out. Report your findings.
      1. The openSLAM project is a repository for various different types of SLAM algorithms composed by different people for different purposes. They provide the source code for the algorithms as well as build guides for compiling and using the different software.
      2. UFastSLAM (unscented FastSLAM) is a variant of SLAM provided by the OpenSLAM project that uses an “unscented particle filter” to provide localization and mapping in a given environment. It overcomes some limitations imposed by the approximations made in a typical FastSLAM by directly operating on linear relations between particles. It is provided in the form of an assortment of MATLAB files that perform the algorithm in simulation. When the provided uslam.m is run, it provides a window that illustrates the movement of a vehicle in a particle field (shown in red), along with a visualization of the linear relations to each of the different particles as it travels (shown in green).  
           
         Figure 2: UFastSLAM output window
   3. Q1.3. 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?
      1. A quaternion is a 4-dimensional number that can be used in the representation of 3D orientation in space. They are used to simplify the rotation of objects in 3D-space and eliminate the problem of gimbal lock, which is the case where a degree of freedom is lost when the object is aligned with an axis. In contrast, Euler’s angles are a representation of 3D orientation with respect to the axes of 3D-space.
      2. The pose and laser scans are transformed between the device frame and the global frame by using a double-multiplication of quaternion rotation vectors:  
           
         where is the device pose quaternion
   4. Q1.4. 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.
      1. Our pure pursuit algorithm begins by taking a vector of waypoints relative to the global origin. The first waypoint in the vector becomes the initial goal for the car. The car then sets the velocity indicated by the goal waypoint and sets the steering angle proportional to:  
           
         where is the straight-line distance to the waypoint and is the distance component in the forward-facing direction.
      2. The car then proceeds toward the goal point. As the car frame then deviates from the global frame, the waypoints are translated by the inverse of the car’s current position and rotated by the negative of the car’s current heading using:  
           
           
         in the form of a rotation matrix [1].
      3. Once the goal point distance reaches a look-ahead distance threshold, the algorithm advances the goal to the next waypoint, and the velocity and steering angle are then set accordingly. This process is repeated until the last waypoint is reached, at which point the goal is set back to the first waypoint.
   5. Q1.5. How does your pure pursuit algorithm perform in comparison to the gap following method?
      1. Our pure pursuit algorithm offers several key advantages, as well as a few disadvantages. The gap follow method that we used previously was limited in that the real-world data coming from the LIDAR was often not as clean as what we had designed around in the simulator. What this meant in practice was that the track would need to be specially designed in order for the algorithm to have good performance, since some deep gaps in the track (i.e. under stools, tables, etc.) would not be the correct path to complete the course. The pure pursuit method solves this problem by knowing the patterns of the track ahead of time and ensuring that the system is not deceived by incorrect data coming in on-the-fly.
      2. The main disadvantage of our pure pursuit implementation was that our feedback for the algorithm is coming entirely from the data provided by the wheel encoders. This means that, over time, small sources of error in the positional data build up to create large offsets in the car’s perceived position vs its actual position, and the algorithm ultimately becomes unstable and crashes. This can potentially be alleviated by augmenting the data coming from the wheel encoders with sensor data from the LIDAR by using SLAM, but this will need to be investigated in the future.
   6. Q1.6. 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?
      1. The first and second columns are x and y position values, the third column is the heading given as an euler angle, and the final column is speed. The position values are in the global frame (i.e. the device frame at the time that the program was started), and the heading and speed are also measured relative to the global frame. If an entry has a zero in column 4, it means that the vehicle came to a stop at that point while the waypoint\_logger was running. We designed our waypoints using a custom MATLAB script as we described in part 1C of our procedures.
   7. Q1.7. What difficulties and problems did you encounter in Lab 6 part 1? What did you try to solve them? What did you learn from this experience?
      1. When we were first developing the pure pursuit algorithm, we had initially considered using quaternion rotation to rotate the waypoints from the global frame into the device frame. While we were doing this, however, we realized that it was going to be somewhat complicated in terms of processing power, and we also noted that all of the rotations will be about the same axis (i.e. the one facing toward the roof of the car) so gimbal lock would never be a problem. Instead, we opted for simpler 2-D rotation matrices, which reduced our rotation calculation down to one line in Python and reduced the processing power needed.
      2. We also encountered some issues while attempting to use our MATLAB script to generate waypoints. After the waypoints were entered, they became offset once we viewed them in RViZ and could not be used to correctly navigate the car. We realized that, in some cases, the origin of the map that we were using as dictated by <some map>.yaml was not compatible with the way we were doing the map overlay in the MATLAB interface. Once we corrected the origin by comparing to RViZ and offsetting the values, the problem was resolved and the program worked as intended.
2. Part 2
   1. Q2.1. What is the Google Cartographer? How is the cartographer implemented in ROS? (Optional) If you tried to modify the shell script for installing the google cartographer, how do you like about modifying the shell script?
      1. Google Cartographer is a SLAM package that is built to be flexible with many different sensor configurations. It has a local SLAM component that generates submaps and the global SLAM component that attempts to align and converge those submaps. In ROS, Cartographer is compiled as a separate package and is able to be used in two different ways. First, maps can be generated live by running Cartographer along with teleop on the car. Second rosbag files can be post processed with Cartographer to generate maps. We did not attempt to modify the shell script for installing Cartographer.
   2. Q2.2. (For ECE450 Students) When you run the demo bag in the cartographer ROS package, what bag file did you use? How long did it take to run the demo? What is the result of the demo?
      1. This was not attempted as the ECE 450 student is remote and has a hard time accessing Cartographer on the jetson.
   3. Q2.3. What was your experience in configuring the car for the google cartographer? What does the f110\_2d.lua do in the package? Hint: check this link and open the file to see the contents. (Optional) If you tried to use the racecar\_cartographer\_install.sh shell script, what did you modify? Submit the modified shell script if you tried.
      1. F110\_2d.lua sets up the initial state of Cartographer. This tells the program which topics to subscribe to and which sensors to use while performing SLAM.
      2. Modifying the shell script was not attempted.
   4. Q2.4. How did you map the race track (SSH or VNC, with rosbag or not)? How many laps did you run and how long did it take? Describe your approach and show your results.
      1. We opted to use VNC/Remmina to start and use Cartographer. However, the procedures outlined in the lab for creating a startup application did not work. Instead, we wrote a simple startup shell script that runs the command and saved it to the home directory. Starting vnc is as simple as starting the robot with a monitor and running that single script. This script may be able to be started using ssh, but we did not test that.
      2. To begin mapping, we discovered that we need to mark our origin to be able to repeat our starting position exactly. Masking tape and alignment using the tiles on the floor was sufficient.
      3. We discovered that driving too quickly while mapping resulted in poor quality maps. Joy\_teleop.yaml was adjusted to limit the car’s top speed to 1m/s. We attempted to drive as smoothly as possible while mapping.
      4. We mapped both the oval test track and the race 2 track. We found that 2 laps around the track was usually sufficient to get a good map. Our reasoning for this conclusion is that we ran 2 laps around the race 2 track, saved a map, and then continued another 2 laps. In comparing the two maps, we observed no consequential differences.

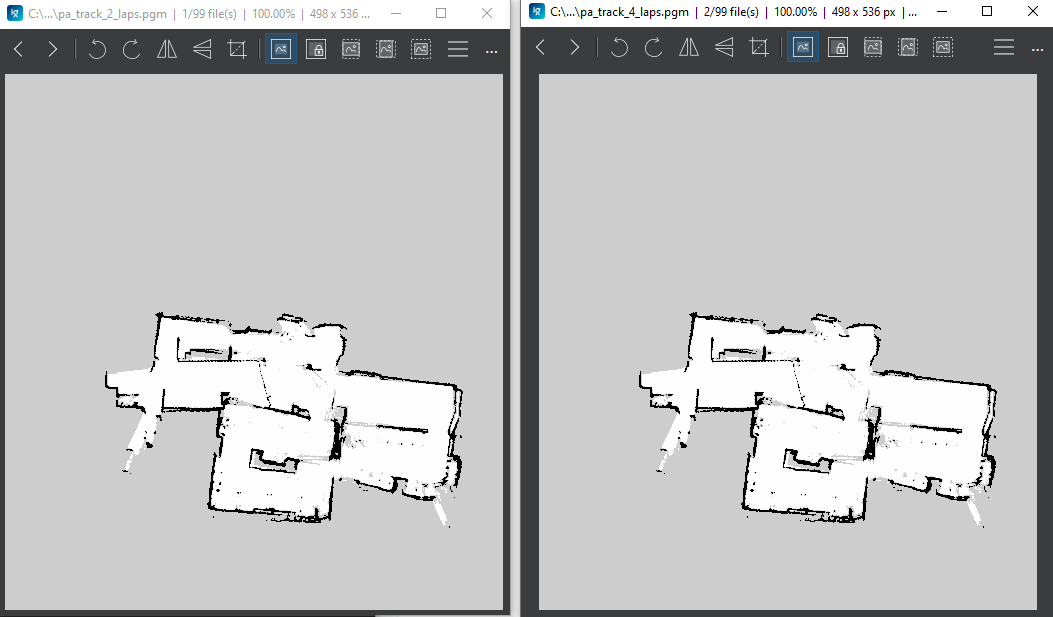


Figure 3: 2 Laps vs 4 Laps

* + 1. While mapping, we observed that Cartographer could not keep up. After completing two laps, we found it best to leave the car stationary with Cartographer still running. The SLAM algorithm will continue to analyze the collected data and continue to align the submaps. After two laps, we found the car needed between 5-10 minutes stationary to fully complete its alignment.
  1. Q2.5. What are the processing methods you used on the map created by cartographer? How did you design your waypoints for your pure pursuit algorithm on the race track?
     1. We used the convert map function to convert the maps to just black and white. We wrote a custom MATLAB script to allow us to import maps and click to set waypoints. We tried to put points on the racing line.
  2. Q2.6. What do you have to modify your pure pursuit algorithm when you run the node on your car vs. on the simulator?
     1. As usual, we had to turn up Kp on the algorithm for it to run on the car. Additionally, we turned down the lookahead distance because the tracks we are driving on in real life are much smaller in scale compared to the maps given in the simulator.
  3. Q2.7. What difficulties and problems did you encounter in Lab 6 part 2? What did you try to solve them? What did you learn from this experience?
     1. A big challenge of ours was that the perceived heading from Odometry was very wrong. When we would perform a 180deg turn, the car would only perceive that it had turned ~85% of that distance. We recalibrated steering\_angle\_to\_servo\_gain in vesc.yaml until we had an appropriate calibration. This value has no impact on the car’s turning radius and only serves to calibrate the encoder distance read by the vesc.

**Conclusion**

1. Using Google Cartographer we were able to drive around real life tracks and generate maps of them using our lidar scans and Cartographer’s SLAM algorithm. We developed a workflow to generate these maps, design points using our custom MATLAB script and load these points into our pure pursuit algorithm. While running the entire Race 2 track has thus far proven to be challenging, we were able to demonstrate initial promising results in the simulator and on the oval test track in the lab room.

**References**

1. <https://matthew-brett.github.io/teaching/rotation_2d.html>