ECE 350/450 Intro to Robotics

**Race 2**

Nicholas LaRosa and Nolan Sornson

Department of Electrical & Computer Engineering

Lehigh University

{nil318, nls221}@lehigh.edu

Date: 4/20/2021

**Introduction**

This report details our second race in the F1Tenth series, this time we are racing with the real car. Our race was performed on a track setup in Packard utilizing portions of PA 227, 230, 228 and 225 (Figure 1). For this race, we decided to use our Pure Pursuit algorithm and localize using Particle Filter on a map we generated with Google Cartographer. We initially struggled with this track because we were unsure how to combine Pure Pursuit and a localization algorithm. Using Pure Pursuit and dead reckoning our position based solely on VESC Odometry data was difficult, we observed that our car would accumulate error quickly and running a long track such as this would prove impossible. However, we soon realized that the Particle Filter program provided to us was a way to eliminate that error by allowing us to correct our pose while driving based on the map we generated. By closing the feedback loop on our Pure Pursuit algorithm, we were able to run our car at high speeds and consistently drive at the limits of the track to minimize our lap times.

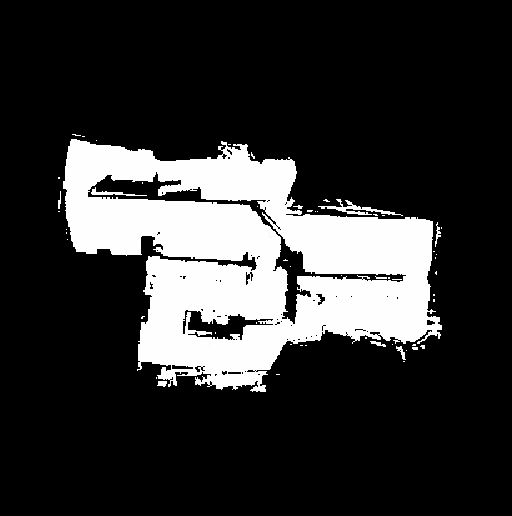


Figure 1: Map of Track Generated using Google Cartographer

**VESC Calibration**

Initially we followed the MUSHR tuning guide to calibrate our VESC; however, we quickly realized that it did not translate exactly to our car platform. While the tuning methods described to set the steering offset and speed\_to\_erpm\_gain are correct, we made some tweaks to the Steering Angle Gain procedures.

* We believe that the 1/10 scale RC that was used in the guide was a slightly different platform than our Traxxas F1/10 Ford Fiesta car. As a result, they calculated the turning radius with a wheelbase of 0.3m instead of 0.324m. Using their half circle estimation, this means we are actually aiming for a slightly larger turning radius.
* We realized that changing steering\_angle\_to\_servo\_gain had little effect on the turning radius, but had a large effect on the car’s perceived pose. When we started tuning, we initially observed that performing a 180deg turn corresponded to a perceived change in pose of only 85% of that rotation. To correctly tune this, we modified a navigation program to set the steering angle to always be maximum and speed to be 1m/s. Then we would let the car run autonomously until it completed the half circle and was facing the opposite direction that it started. We could then print out the perceived heading in the terminal and tune steering\_angle\_to\_servo gain by aiming for a target of -3.14 radians. While we were able to tune these values and get them dialed in as best as we could, collecting dead reckoning data from the VESC still seemed to accumulate error quickly.

**Google Cartographer**

We struggled to make maps using Cartographer for the majority of the time spent on pure pursuit. Here are some of the considerations we had in mind while we were mapping in order to get the best results.

* The car needs to run slow. We modified joy\_teleop.yaml to make the top speed in teleop be 0.6m/s.
* The VESC tuning needs to be decent. Both speed\_to\_erpm\_gain and steering\_angle\_to\_servo\_gain need to be close to the correct values, but you will be able to get away with it even if they are not perfect.
* Turn on use imu data in f110\_2d.lua. This configuration file has this initially set to false. Because our VESC tuning was not perfect, we think the IMU data from the VESC helped make a more accurate map. (Figure 2)



Figure 2: Map with IMU data on vs Map with IMU Data off

* Run two laps continuously and then let the car sit. Two laps was the sweet spot in terms of data collection for us, and Cartographer needed to sit still at the end to match scans and get rid of erroneous data. We found that if our submaps were askew they would usually converge within ~5 minutes.
* Clean up converted pgm files in GIMP. We found that, no matter how well we drove, there would be some erroneous edges that appeared on our final maps (Figure 3). These would throw off the Particle Filter program later, so we chose to edit them out using the pixel brush in GIMP.

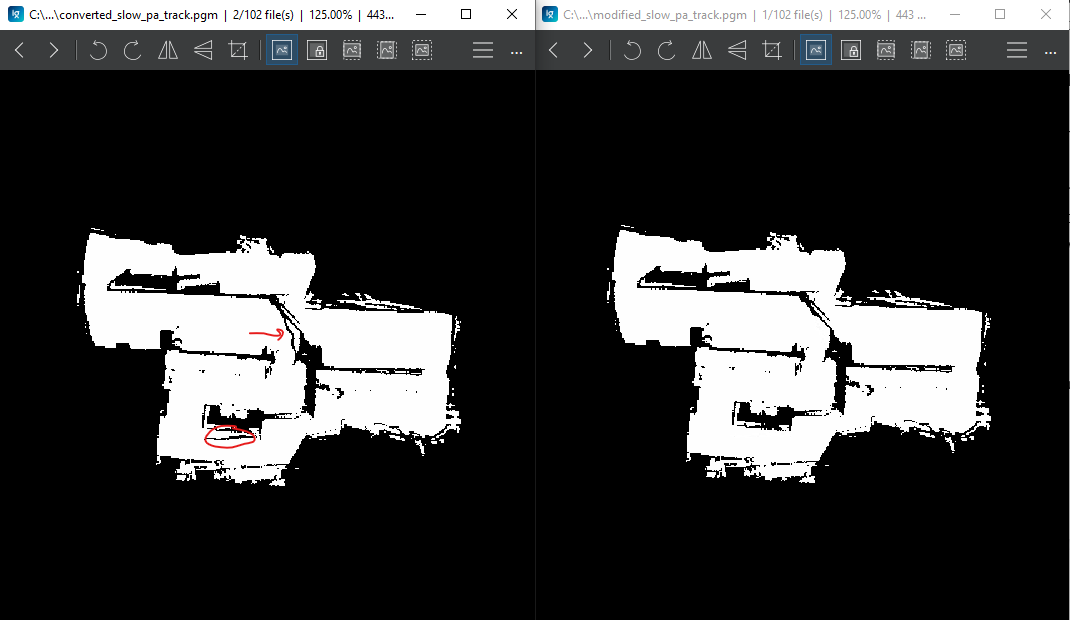


Figure 3: Converted Map from Cartographer vs Map Modified in GIMP

**Pure Pursuit + Localization**

The algorithm that we decided to use on our car is both a preemptive and reactive method based on Pure Pursuit with data augmented using particle filter localization. First, the desired path of the car in the form of a set of waypoint coordinates and speeds is fed to the algorithm ahead of time in the form of a CSV file. The algorithm starts by setting the speed to the value indicated by the first waypoint. As the car begins moving and the position begins to change, the waypoint map is translated and rotated into the car’s frame by taking the current pose and translating and rotating each point by the inverse of the car’s pose (similar to the method used in Lab 6):

Figure 4: Rotation matrix equations

Once the entire waypoint array is in the car frame, the steering angle is set proportionally to the result of the arc calculation:

where is the straight-line distance to the waypoint and is the distance component in the forward-facing direction.

Our approach to the look-ahead distance was slightly different from the method used in the slides. Instead of looking for waypoints within our lookahead distance bubble, it looks for the first waypoint in the list that is outside the bubble and starts driving toward it. Once the point reaches the lookahead distance threshold, the focus of the algorithm advances to the next point, and the process repeats. This eliminates any issue that may arise from waypoints being spaced too far apart, and it also tends to increase the stability of the algorithm by avoiding aggressive turning maneuvers trying to reach points that are very close. In addition, we added another condition that tells the algorithm to advance to the next waypoint if the current one is behind it. This corrected a sporadic issue where the car would turn around to reach a point it thinks it missed. Adding this correction solved the problem under most conditions.

In the simulator, it was possible to run this algorithm with good performance under the simulator since it did not account for errors in the odometry data and was always accurate. In practice on the physical car, however, this is not sufficient; incremental errors in the data due to sliding and sensor inaccuracy quickly generate a building offset in the odometry data that must be corrected from another source. To do this, we decided to run the particle filter algorithm provided in [2] along with our pure pursuit. The particle filter takes the odometry data coming from the wheel sensors and generates a set of particles corresponding to the expected LIDAR data at that point. It then takes the actual LIDAR data, compares the particles of the two, and probabilistically calculates and publishes the correct position to /pf/vis/inferred\_pose based on the difference between the particle maps. Once this is done, we finally simply subscribe to the inferred pose data and use this instead of the odometry data. In our testing, this method works extremely well to correct the odometry data, and it allowed the pure pursuit algorithm to continue to work in the presence of odometry errors.

**Making Waypoints and Tuning**

In order to collect the waypoints to be used in our race algorithm, we created an additional MATLAB script that takes the Cartographer map as an input and allows the user to graphically select waypoints on the map to be used using the cursor. Once the input is complete, the script then graphically illustrates the input and generates a 3-column CSV file indicating the x and y positions of each of the waypoints selected, along with a third column containing a placeholder speed value for each point.

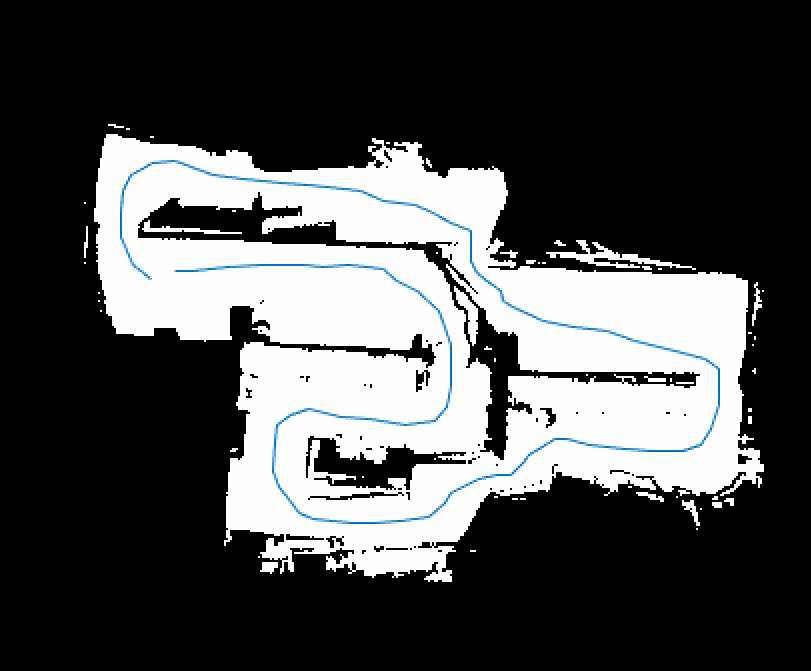


Figure 5: Illustration of points overlaid on map, generated by MATLAB

Once the waypoints are generated, we then open the file in Excel to fine-tune the waypoints and add a speed value target for each of them. This is done by running our pure pursuit algorithm both on the simulator and later on the car and using trial-and-error to determine the maximum possible speed and optimal placement of the waypoints.

There are a few issues we ran into as well as a few improvements that could be made to the workflow. When we originally tried importing the map and overlaying the points, we found that they were all offset, sometimes significantly, from their expected positions on the map. The cause of the issue appears to be either that the map origin specified in the YAML file for the map is incorrect, or the way that MATLAB defines the origin is different from the way that ROS does. In either case, we corrected the issue by manually modifying the MATLAB origin until the points were placed correctly. Another issue is that the individual modification of the points and adding the speed values in Excel can sometimes take a long time, especially if there are a lot of points. A future improvement to the process could be to design an interface, similar to other SLAM projects, that allows us to easily move and modify the properties of each point individually. With a tool like this, it would be much easier to optimize the waypoints to reduce the needed lap time.

**Results / Analysis**

Using Pure Pursuit and localizing while driving using Particle Filter, we were able to achieve very quick lap times. Our first run during the race was close to 22 seconds, which was exactly the time we measured the previous night during our testing. For our second and third runs, we focused on increasing the minimum cornering speed and increasing the time we spent at our top speed of 4-4.5m/s and were able to achieve lap times of 20.96s and 20.60s respectively.

**Conclusion**

By using both our Pure Pursuit Algorithm that developed for Lab 6 and the Particle Filter program provided to us, we were able to design a set of waypoints for a previously mapped track and have our car follow them consistently and at a high speed. While our VESC tuning was not perfect, localization using Particle Filter helped us correct the pose as VESC Odometry error accumulated, ensuring our car followed the waypoints accurately. Including speed in our waypoints csv file decreased the time it took for us to modify the speed throughout every point during a lap. Decreasing this tuning time allowed us to push changes to the car rapidly and then test then right away to continue improving our lap times. These decisions allowed us to attempt many full laps on the real track which ensured we could achieve an optimal lap time.

**References**

1. <https://matthew-brett.github.io/teaching/rotation_2d.html>
2. <https://github.com/f1tenth/particle_filter.git>