Benjamin Grudzien

Professor Louie

Introduction to Autonomous Vehicle Systems

18 March 2020

Autonomous GPS Waypoint Navigation Project

My GPS simulation project has a simple and straightforward structure. After running 30 tests; I concluded this algorithm has an 86% success rate. The average time is 45s will an average variation of 15ms. Of those 30 test, 8 of them finished in under 45 seconds. The highest rate of failure occurs between waypoint 7 and waypoint 8 as I pushed the car to its limits to achieve an under 45 second time. It's important to note that 3 of my failures occurred in a row towards the end of my testing; tests 26, 27, and 28. I then cleared my cache, rebuilt my program, and ran 4 straight passing simulations. Perhaps this suggests errors can occur more frequently as the cache builds up, or maybe that is my own bias. Another important thing you should know is that I named my variables concisely and added plenty of comments to my code to aid your understanding and increase its readability.

From a high-level, my program works like this. I have one node which is **gps_sim_project**. This node subscribes to **audibot/gps/heading** and the **audibot/gps/fix**. This gives me information about the current vehicle heading angle and the current vehicle location respectively. My algorithm publishes to **/tf, gps/path, /marker,** and **audibot/cmd_vel.** This allows me to view the markers and gps path in rviz, and control the speed/direction of the car. An overview of this configuration is shown in Figure 1.

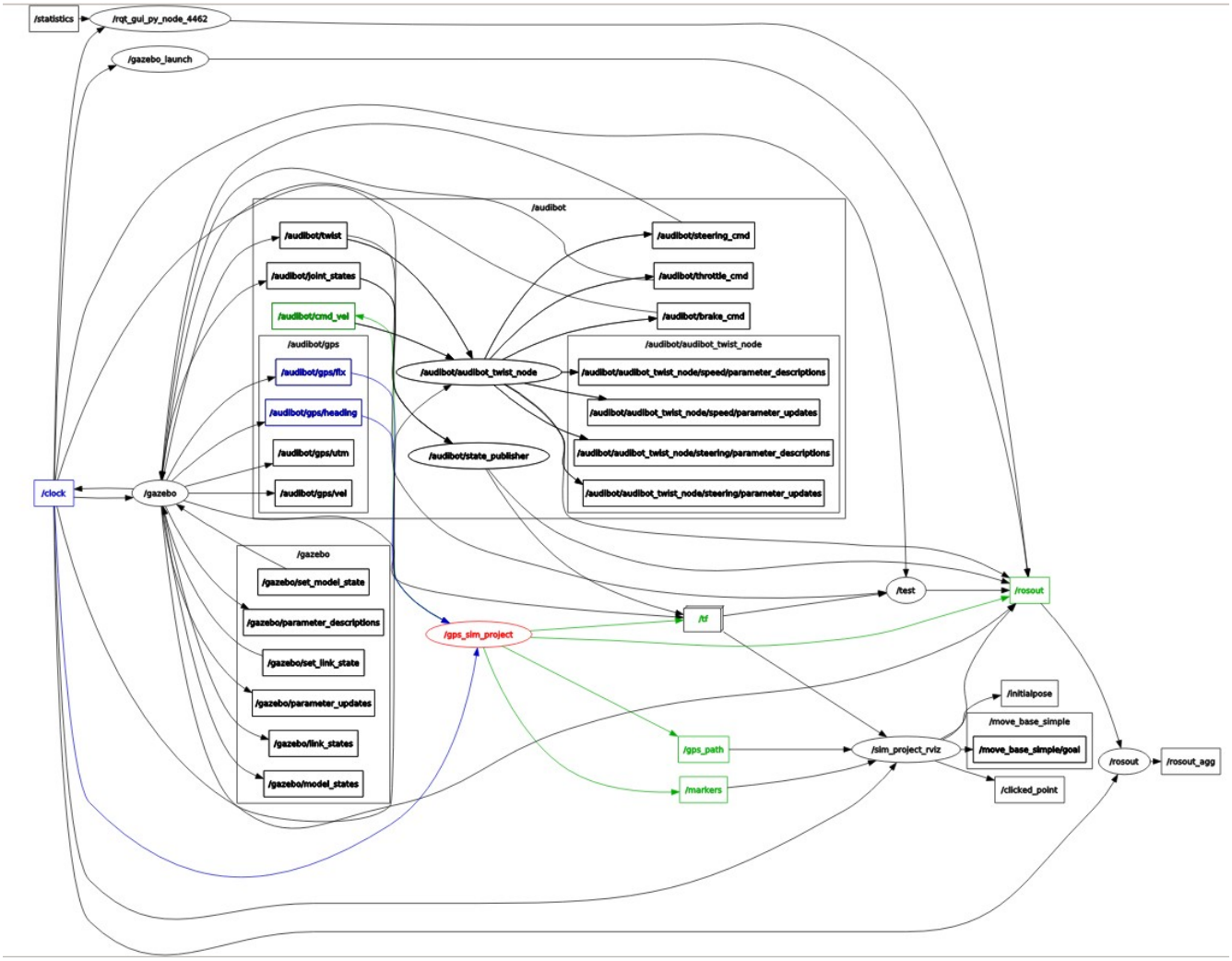

Figure 1: rqt_graph screens of my project (This picture is also saved in this folder).

The program is straightforward. First, I initialized the waypoint locations by creating a vector between them and the starting coordinate in UTM. I got the starting coordinate by running a ROS_INFO() and printing out the starting coordinates to the screen. I got this from my **recvLocation** function. This function also gives me the current position of the car. Next, I received the heading error of the vehicle relative to true north in the **recvHeading** function. After that, in the **displayCallback** function, I very simply set up the markers and the gps path to display in rviz. The meat of my program occurs in the **algo_Callback** function. This is where my algorithm lives. First, my algorithm collects a lot of data and updates all my variables. It collects the current car location, creates a vector between the

car and the vehicle, finds the angle between that vector and UTM north, finds the convergence angle between UTM north and true north, and then finds the angle between the vehicle heading and the marker accounting for the convergence angle. After this information is collected. My algorithm checks to see what waypoint is hit. The waypoint status is stored as an integer and that is used to cycle through the following switch/case statements. At a high-level, these case statements tell the car to go straight, left, or right depending on the angle between the car and the marker using if-statements. These if-statements tell the car to drive a certain speed, or turn at a certain rate. Note, there are some outlier situations in those case-statements based on distance to the waypoint. Finally, my program enters the **main** function which basically just calls all the functions above in an infinite loop.

In conclusion, I had a lot of fun doing this project and I could see myself doing this when I graduate. Time absolutely flew by programming this, especially when I was tuning the algorithm to hit the markers faster and more consistently. If I could do this project again however, I probably would have implemented a proportional controller that accounted for the distance to the next waypoint and the angle to the waypoint to calculate the desired speed and turning rate. I think that would have been a more universal and “autonomous” way of doing this. Nonetheless, I am still happy with my results.