ECE 350/450 Intro to Robotics, Lab 3

**Wall Following**

Nicholas LaRosa and Nolan Sornson

Department of Electrical & Computer Engineering

Lehigh University

{nil318, nls221}@lehigh.edu

Date: 3/23/2021

**Abstract**

In this lab, we designed and implemented our first level 2 automation algorithm: wall following and speed control using LIDAR. Our algorithm uses a PID (proportional-integral-derivative) controller to maintain a set distance from the left wall of a track, and it also implements a simpler algorithm to moderate the velocity as a function of the steering angle to increase the maximum speed. We first tested the algorithm using the F1Tenth simulator, which allowed us to perform most of the algorithm design before working on the car. After transferring the algorithm onto the Jetson board on the car, only a few parameters needed to be adjusted before the algorithm could be run on the car hardware. This report details our procedures and results.

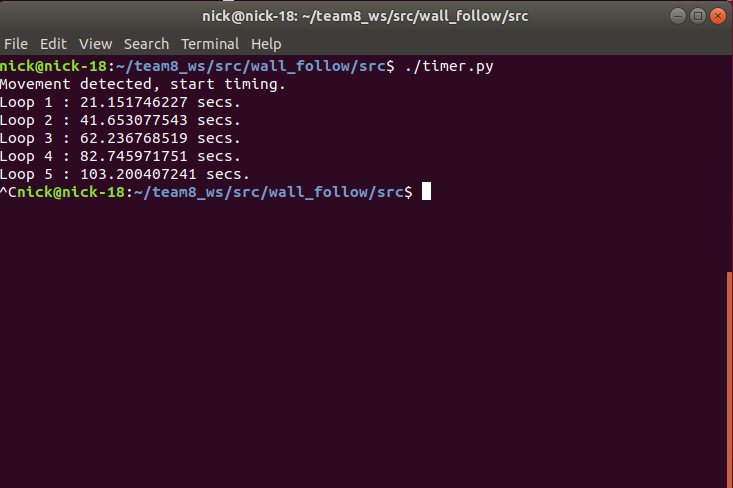
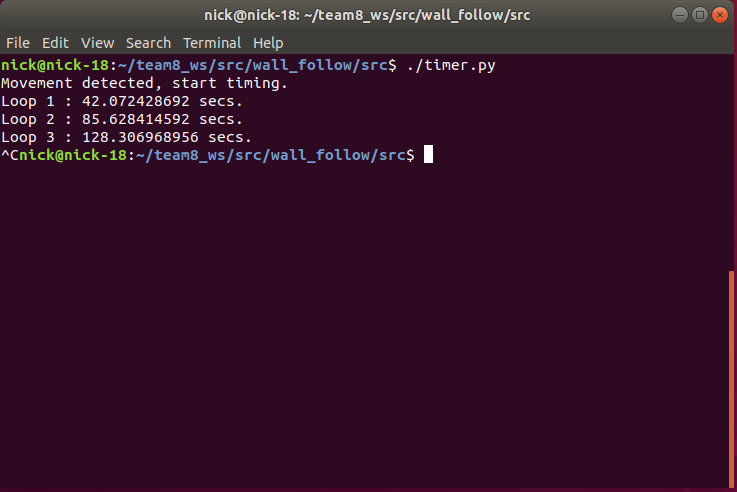
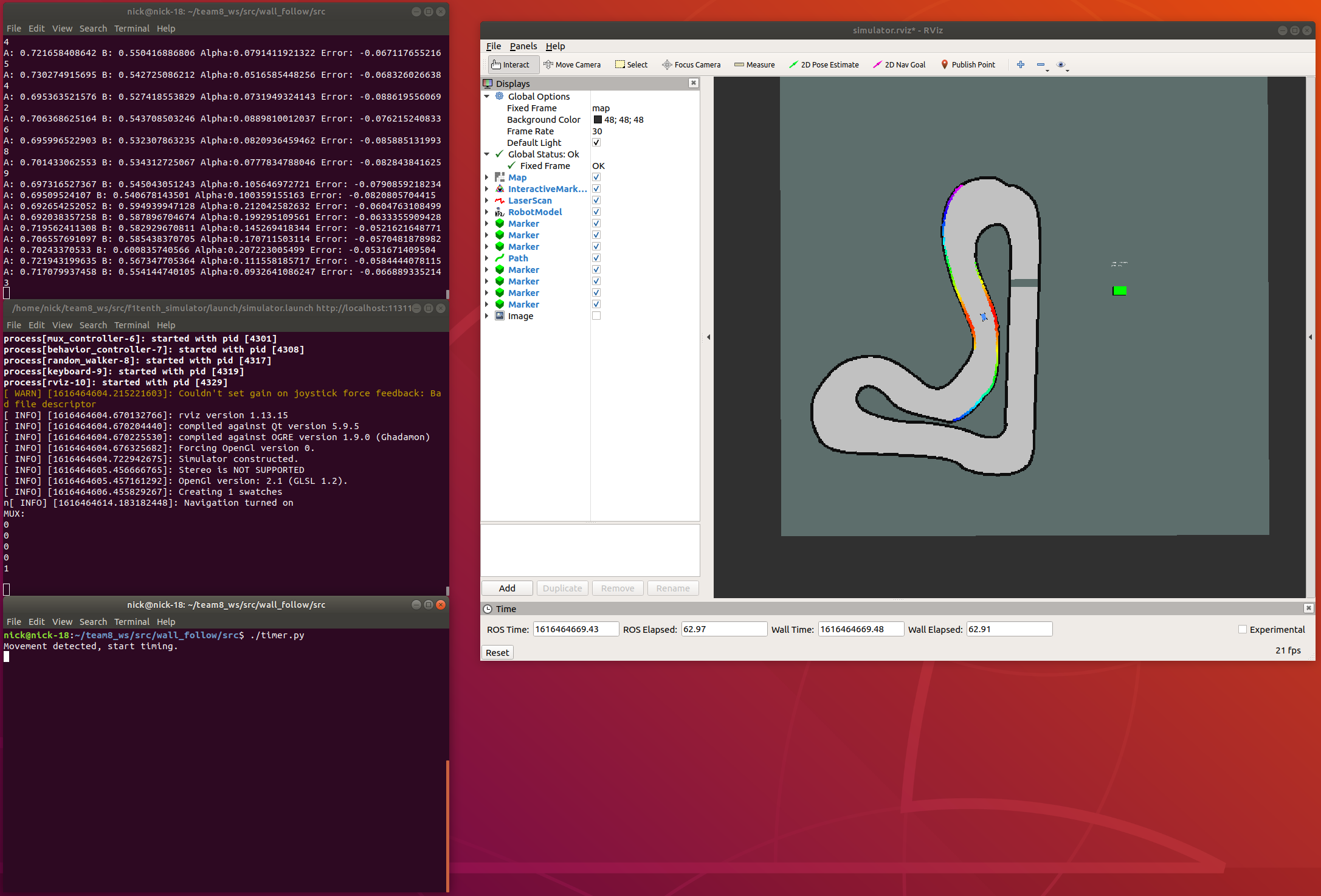
**Introduction**

1. Part 1
   1. LIDAR - a method for measuring distance and range using the time it takes for a laser to be reflected back. Using the data, a 3D representation of the area seen by the sensor can be constructed. The LIDAR module used in this class is the Hokuyo UST-10LX which has a detection range of 10 meters and 1081 measurements per scan over a field of view of 270 degrees.
   2. Navigation Control via keyboard - a control method for the simulated and actual car that uses the keyboard and mouse to generate ROS messages and operate the vehicle motors and steering
   3. PID Control - Acronym for “proportional-integral-derivative control” a control method that takes an error value and produces a correction that depends proportionally on the error, the derivative of the error, and the integral (accumulation) of the error. It produces a second-order control system to achieve quicker and more reliable convergence when tuned.
   4. Look ahead distance - The distance that the LIDAR “looks ahead” along the wall to determine the steering correction error value to send to the PID controller.
      1. Determined by and , whose difference is approximately 60-70 degrees
      2. Allows us to make pre-emptive corrections to the steering angle to improve the PID controller performance and increase the maximum speed.
2. Part 2
   1. SSH - Secure Shell is a network protocol for remotely accessing computers on the same network. Headless computers such as servers or robots can be accessed using public key encryption which provides a secure connection, even if on an unsecured network. SSH uses port 22 to connect with.
      1. SFTP - Secure File Transfer Protocol is an extension of the SSH protocol that allows for files to be passed between computers remotely connected with an SSH tunnel.
   2. Udev
      1. This system identifies devices connected to a linux computer. In normal operation, udev will assign devices randomly within the operating system as they are plugged in, but if the user wants more customization, udev rules can be written to define what names to assign or actions to take upon add device being connected.
   3. .yaml - Stands for YAML Ain’t Markup Language, YAML is a data format that aims to be less verbose than XML. Values can be stored in a YAML file and loaded as initial conditions for a program.
   4. .xml - Stands for Extensible Markup Language, these files are both human and machine readable and allow for scripts for complex low level operations to be performed.
3. Part 3
   1. Teleop - Tele-Operator mode is activated in ROS by holding LB on the controller during operation. LB acts as a dead-man's switch and commands can only be sent to the car when LB is active. In the event of errors and issues, LB can be released and the car will stop
   2. Autonomous navigation via controller - autonomous navigation mode is activated in ROS by holding RB on the controller. This button is also a dead-man’s switch so no commands can be sent to the car when the button is not pressed, preventing the car from being stuck in a dangerous loop.

**Procedures**

1. Part 1
   1. For the first part, we started by creating a new catkin package “wall\_follow” in our team workspace, similar to the way we created the AEB package. We then copied the Python file from Lab 3 of f1tenth\_labs and used this file as a launching point to develop our algorithm.
   2. The wall\_follow node subscribes to the /scan and /odom topics and includes one AckermannDriveStamped publisher to the /nav topic.
      1. Upon receiving a new data set from the LIDAR on the /scan topic, the algorithm clips the data in each range array element to the minimum and maximum produced by the sensor. This is essentially the same as what we did for the AEB implementation. Next, a second array is constructed with the same dimensions as the filtered range data, and it is filled with the angles corresponding to each element in the range array.
      2. The next step is to find the ranges corresponding to the angles and To do this, we search for the index of the angle array that is closest to the desired angle, and, after finding that index, get the value in the corresponding index of the range array. This provides us with two vectors “a” and “b”
      3. We then use trigonometry to find the offset angle using the formula . This angle is used to derive the distance from the wall in the next time interval and consequently . This error signal is then fed to the PID controller to generate the steering angle based on the formula , where and are the PID controller coefficients. The desired steering angle is then published to the /nav topic to make the change on the simulated hardware.
   3. To run the algorithm on the simulator, we imported a new map that is closer to a physical track and added it to the launch file. Finally, we further modified the simulator.launch file to include the python file for our wall follow algorithm, which would then run on the simulated car by pressing “n” on the keyboard.
2. Part 2
   1. The team began by cloning the f1tenth\_system directory into the src folder of our workspace. While this initially seemed like the right step, the team quickly realized that f1tenth\_system was not actually the “package” that ros wanted to see. As a result, any package relative commands (e.g. roslaunch <package> <.launch file>) would not autocomplete for f1tenth\_system. While typing out the full name for f1tenth\_system would allow the next argument to be automatically completed, the team decided to copy the contents of f1tenth\_system into the src directory. $cp -r \* .. was a quick way to move all of the contents from the current directory, up one level. Now all of the packages in f1tenth system were all able to be autocomplete when using the ROS package relative CLI commands.
   2. Initial errors arose when catkin\_make was first performed in this workspace, but this was quickly resolved by reading the f1tenth Workspace Setup documentation [1] and realizing that ros-melodic-driver-base needed to be installed from apt. This documentation all provided a nice command to recursively find all python scripts and make them executable, ($find . -name “\*.py” -exec chmod +x {} \;) which saved a lot of time.
   3. Udev rules were installed by downloading files from coursesite and moving them to /etc/udev/rules.d. After reboot, the team did not initially see the rules activate, but realized that the VESC needed to be powered for the VESC rule to appear.
   4. All yaml files were modified as directed. Sensors.launch.xml was also modified to start the urg\_node for the LIDAR. This is a really key step to understand, because this xml file is called during teleop.launch to start the LIDAR. A custom launch file for the wall\_follow node does not need to also start the lidar, which was a bug that stumped us for a while later on.
   5. The car can be run with the joystick by plugging in the usb receiver, turning on the controller by pressing the logitech button and then the mode button until the green light is on.
   6. Headless operation mode can be achieved by using the battery to power both the VESC and the Jetson. Using the host computer, ssh into the Jetson and launch the teleop program.
3. Part 3
   1. The team copied wall\_follow.py from the host computer to the Jetson using SFTP. The team initially had a few missteps trying to get the autonomous wall following program running.
      1. First, the team wrote a custom launch file to start the car, the lidar, and the wall following node. However, the team forgot that the lidar was already started by the sensors.launch.xml and starting it twice actually caused the VESC to crash. When the launch file was modified to just start the car and the wall following node, no errors were thrown.
      2. Second the team had correctly modified the drive topic on an earlier version of wall\_following but then overwrote that file with a newer version that was tested in the simulator. While the car started with no errors, no movement was observed when RB was held. This was finally resolved and the car was able to run on the bench in autonomous navigation mode.
      3. Third, when transitioning to headless mode to drive on the ground, the VESC usb connector was bumped and actually disconnected from ROS. A reboot was required to resign that device before driving on the ground.
   2. The car was first tested with the initial PID parameters that were developed in the simulator. However, the car was nowhere close to being able to make the turns on either end of the track. The Kp value was increased drastically until the car was able to drive around the track once at 1 m/s. Then the speed was increased and Kd adjusted as oscillations arose. Finally values that proportionally reduced the car’s velocity as the steering angle increased were adjusted until the car could run at 4m/s down the straights, but drive slower on the turns to ensure that it would not hit the sides. This allowed us to achieve ~7 seconds a lap when using hand timing.

**Results and Analysis**

1. Part 1
   1. Q1.1. How does the PID control work for the wall follow algorithm? How do you select the parameters of proportional, integral, and derivative control methods?
      1. The PID controller takes the error signal, which is derived from the vectors a and b and its corresponding angle . This is then used, along with the set velocity, to determine the predicted distance from the wall in the next time increment. The difference between the predicted distance and the desired distance becomes the error supplied to the PID controller, which then outputs the steering angle using the function:
      2. We started by setting and to zero and only adjusting . We increased until the system became responsive enough to avoid hitting the wall. Then, we increased to eliminate the steady-state error, and then increased to dampen out any steering oscillations. After the rough tuning was complete, trial and error was used to fine-tune the values to obtain optimum performance.
   2. Q 1.2. How would the PID control parameters change when you set a different velocity?
      1. In our experience, the PID parameters did not themselves need to be changed when the velocity was changed, since the velocity was a component of the discrete integration and differentiation of the PID algorithm. However, in practice increasing the velocity did sometimes expose incorrect tuning that remained latent when the car was slower (mainly oscillations) and it sometimes required additional adjustment of the parameters.
   3. Q1.3. How did you test your WallFollow node in the simulator? How did the parameters affect the results? Show the test results with two different speeds of the race car.
      1. To test the wall follow algorithm in the simulator, we created a new launch file based on simulator.launch and included the wall\_follow.py file. We also ran the timer.py file from the coursesite to determine the lap time with different parameters and velocities. The PID parameters usually did not have a significant effect on the lap time (unless there were significant oscillations that did not result in a crash), but the car velocity and turn factor parameters played a significant role on the final lap time.  
           
           
           
         *Figure 1: Output of timer.py for velocity=3.25, turn factor=4  
           
           
         Figure 2: Output of timer.py for velocity=2, turn factor=0  
           
         Figure 3: Example of wall follow algorithm running on simulated car*
   4. Q 1.4. (ECE450 Students, option 1) Derive the Laplace transfer function for the PID controller. Comment on how the parameters can cause the system to be unstable. Make your own assumption about the plant being controlled.  
      1. The time-domain expression for the PID controller is given by:  
           
         Taking the Laplace transform gives:  
           
           
           
         We will assume that the initial conditions of the error are zero. We also assume that the plant (the car in this case) is stable under all conditions. Multiplying through by gives:  
           
           
           
         This is a second-order system. The transfer function of the system is then given by:  
           
           
           
         The PID controller will be stable as long as the combined transfer function , where is the transfer function of the plant, converges to zero for any . Since we assume that the plant is stable, this means that the overall system is stable as long as is stable. The roots of are given by:  
           
           
           
         If the roots are complex conjugates (i.e. the part under the square root is negative), the system could still be stable given the previous condition, but there will be a decaying oscillation (as observed when is large).
2. Part 2
   1. Q2.1. What is the reason for using the udev rules? How do you activate /trigger the new rules?
      1. Udev rules allow for static assignment of certain devices. In this case, the team used udev rules to assign the lidar and the vesc to the same names and paths each time the jetson is booted up. This simplifies the code that the team needs to write to interact with these devices.
      2. When the rules are first installed, they must:
         1. First: be reloaded (sudo udevadm control --reload-rules)
         2. Second: be triggered (sudo udevadm trigger)
   2. Q2.2. What do you observe when you run the car using the joystick?
      1. When using the teleop mode, the car is rather sensitive to joystick inputs. Both throttle and steerings require very little input on the controller to create a rapid response from the car. Care is required when driving around with the joystick to avoid hitting objects
   3. Q2.3. When running the teleop with SSH, what would ‘tmux’ help you to accomplish?
      1. When using ssh on the host computer, everytime a new terminal is opened, you have to ssh back into the jetson. Using tmux allows you to have multiple terminals within one window, meaning you don’t have login to ssh every time. This helps when booting straight to headless operation because roscore and your launch file must be started in separate terminal windows. Tmux could further be used to display telemetry from the car in a meaningful way.
      2. A quick guide to tmux can be found in [2].
   4. Q2.4. If you modify the key assignment of the joystick, then report your new key assignment and submit your edited yaml file.
      1. This was not attempted because the current teleop setup functions okay.
3. Part 3
   1. Q3.1 How are the wall\_follow parameters different from the ones used in the simulator? How do the parameters affect the behaviour of the car?
      1. There are some differences between the wall\_follow parameters used on the simulator and the car. For one, the physical car seems to have a lower turning radius than the simulated car, which means that tight turns must be made earlier and more sharply. Also, the physical track in the lab is smaller than the simulated one, which changes the requirements for the desired wall distance. To correct for this, on the physical car we changed the desired wall distance to 0.4 meters, increased the PID parameter, and changed the angles and to 15 and 75 degrees from 30 and 90 respectively. The wall distance change brings the car closer to the wall and consequently close to the middle of the smaller track. Changing the tells the car to react more aggressively to sharp curves. Finally, changing the angles as indicated allows the car to look further ahead and act more preemptively to curves. With these changes, our wall follow algorithm was able to run on the physical car.
   2. Q3.2 What do you observe when you run your wall\_follow node with the car? How much time do you need to finish two laps? How do you time the run?
      1. All measurements were performed by taking a video and playing back the footage to see when the car started moving and when it crossed the starting line.
      2. Our first completed lap followed the outside wall and required a target spacing of 0.4m. This was completed at 1m/s and took ~21 seconds to complete one lap.
      3. Increasing the velocity to 2m/s, we cut our time down to 13 seconds.
      4. Increasing the velocity to 4m/s, but decreasing the velocity proportionally to the steering angle reduced our laptime to 8 seconds.
      5. Further improvements can be made by increasing velocity on the straights and increasing the minimum cornering speed.
   3. Q3.3 Run a rqt graph for the teleop.launch and explain what you learn from the rqt\_graph about the racecar package.
      1. After running rqt\_graph we observed a lot of different topics and nodes that are all connected. The node we created, /wall\_follow, talks to the topic /vesc/high\_level=/ackermann\_cmd\_mux/nav\_0. This is located within /vesc/high\_level, which appears to handle all of the computation and input from the joystick. /vesc/high\_level then sends information to /vesc/low\_level which finally passes data to /vesc/commands which set the motor speed and servo position. Both of those last two topics talk to the /vesc/throttle\_interpolator which moves the car and any sensor data is sent to /vesc/vesc\_to\_odom, which completes the feedback loop.

**Conclusion**

In this lab, the team was able to develop a basic wall\_following program in the F1Tenth simulator and then take that program and put it directly into the car with minimal modifications. Through this process we now understand the software development process for an autonomous vehicle in ROS. Being able to develop the algorithm and test it at a high level in the simulator will let the team solve the major bugs, before taking on significant risk by running it on the car. However, the simulator is not one to one with reality and the team should expect to have to tune parameters on the car each time significant changes are made to the software. Our preliminary testing of our wall\_follow program in the simulator allowed us to improve our lap time by 39% when compared to the demo lap of 27 seconds. Our testing of our wall\_follow program on the car itself allowed us to improve our lap time 62% within an hour of testing.

**References**

1. <https://f1tenth.readthedocs.io/en/stable/getting_started/driving/drive_workspace.html#doc-drive-workspace>
2. <https://www.hamvocke.com/blog/a-quick-and-easy-guide-to-tmux/>