ELECENG 3EY4: Electrical System Integration Project

Lab06_Setting Up Simulation Environment and Manual Driving of McMaster AEV

Junbo Wang – wangj430 – 400249823

Preet Batra - batrap5 - 400341704

Yichen Lu - luy191 - 400247938

Activity 1: Installing Simulator Software Stack

For this activity, we basically just followed the guide on the lab manual and the GitHub repository to install the additional package dependencies which are tf2_geometry_msgs, ackermann_msgs, joy, map_server.

Activity 2: Replace the existing “params.yaml” with the one posted on the course website.

For this activity, we replaced this file,called **params.yaml,** with the one with the same name provided to you on Avenue to Learn by executing the following commands:

$ roscd f1tenth_simulator/

$ sudo gedit params.yaml

This ensures that our simulation parameters will more closely reflect the specifications of the McMaster AEV, increasing consistency between your simulations and experiments.

Activity 3: Set the values for the two parameters update_pose_rate and scan_beams in “params.yaml” file

Update_pose_rate is 0.001, which determines the LiDAR scan rate.

Scan_beams is 720, which determines the number of LiDAR returns per one full scan.

These two values are provided by TA in the lab session.

Activity 4: Examine the content of the file “simulator.launch” under “/launch” directory in simulator package.

For this activity, we just typed the command, **sudo gedit /launch/simulator.launch**. Afterwards, we see the exact content that is shown in the lab manual.

Activity 5: What is a node in ROS?

Nodes refer to the execution of computational processes in the ROS system. They communicate with each other by passing messages, publishing messages to topics, and sending requests to other nodes.

Activity 6: Briefly explain the role of a Launch file in ROS. Compare the use of launch file to an alternative where the nodes are run individually by using the ROS command “rosrun”, i.e., $ rosrun package node _parameter:=value

There are many roles of a Launch file in ROS. For instance, a launch file can be used to activate several nodes and master more quickly. Additionally, it also can set multiple parameters.

As for the alternative way, which is run individually by using the ROS command “rosrun”, there are advantages as well. For example, it is more convenient to run compared to the launch file in ROS.

Hence, it is better to use the ROS command “rosrun“ to run some simple tasks. However, for complex applications, it is better to use the ROS Launch file.

Activity 7: Attempt to launch one of the nodes from the list of nodes in the launch file using the “rosrun” command and report on the result.

We used the “rosrun” command to launch one of the nodes from the list of nodes. It worked perfectly.

Activity 8: Briefly explain the role of each node launched by “simulator.launch” and relate them to their corresponding blocks in Fig. 1.

Joy Node (<node pkg="joy" name="joy_node" type="joy_node"/>): This node is responsible for interfacing with a physical joystick. It translates the joystick's input into ROS messages on the /joy topic, which can be used to manually control the simulator. This node corresponds to the "Joystick" block in Fig. 1.

Map Server Node (<node pkg="map_server" name="map_server" type="map_server" args="$(arg map)"/>): This node serves a map from the maps folder to other nodes. It's essential for providing environmental data to the simulator and other nodes. This node corresponds to the "Simulator " block in Fig. 1.

Racecar Model (<include file="$(find f1tenth_simulator)/launch/racecar_model.launch"/>): This node is the launch file for the race car model, which initializes the model of the car used in the simulator. It doesn't correspond to a specific block in Fig. 1 but it is essential for visualizing and stimulating the racecar's behavior.

Simulator Node (<node pkg="f1tenth_simulator" name="f1tenth_simulator" type="simulator" output="screen">): This node starts the actual simulation environment where the vehicle operates. It loads parameters from params.yaml and likely simulates the vehicle's physics and sensor data. This node corresponds to the "Simulator" block in Fig. 1.

Mux Node (<node pkg="f1tenth_simulator" name="mux_controller" type="mux" output="screen">): The Multiplexer (Mux) node chooses which control commands would be sent to the simulator based on priority. This node corresponds to the "Mux" block in Fig. 1.

Behavior Controller Node (<node pkg="f1tenth_simulator" name="behavior_controller" type="behavior_controller" output="screen">): This node acts as the autonomous decision-making system that takes sensor data and calculates the appropriate responses or driving commands. This node corresponds to the "Behavior Controller" block in Fig. 1.

Random Walker Node (<node pkg="f1tenth_simulator" name="random_walker" type="random_walk" output="screen">): The Random Walker node probably simulates random driving behavior for testing purposes. It could provide randomized control commands to the simulator. This node corresponds to the "Random Driver" block in Fig. 1.

Keyboard Node (<node pkg="f1tenth_simulator" name="keyboard" type="keyboard" output="screen">): This node is made for keyboard input which allows control commands to be sent to the simulator through key presses. This node corresponds to the "Keyboard" block in Fig. 1.

New Planner Node: This node would be a custom node for a new path planning algorithm.It is provided in the launch file. The purpose of this node is to plan for new drivers. This node coresponds to"New Planner " and "New Planner2 " blocks in Fig. 1.

RVIZ Node (<node pkg="rviz" type="rviz" name="rviz" args="-d $(find f1tenth_simulator)/launch/simulator.rviz" output="screen"/>): RVIZ is a visualization tool in ROS. This node launches RVIZ with a specific configuration file which allows you to visualize the simulation, including the vehicle, the environment, and sensor data. It does not correspond to any block in Fig. 1 but it is necessary to observe the system's state visually.

Activity 9: Launch the Simulator by executing the following command: $ roslaunch f1tenth_simulator simulator.launch

Launch the Simulator by executing the command:

$ roslaunch f1tenth_simulator simulator.launch

This command will launch several nodes and the “rviz” visualization environment.

Activity 10

For this activity, we firstly stopped the simulation by pressing ctrl-c. And then we opened a terminal window and run the following command:

$ roscore

Afterwards, we opened another terminal window and run the following command:

$ rosparam set enable_statistics true

At last, we re-ran the simulator by launching “simulator.launch” file and executed the command below to run rqt_graph plug-in:

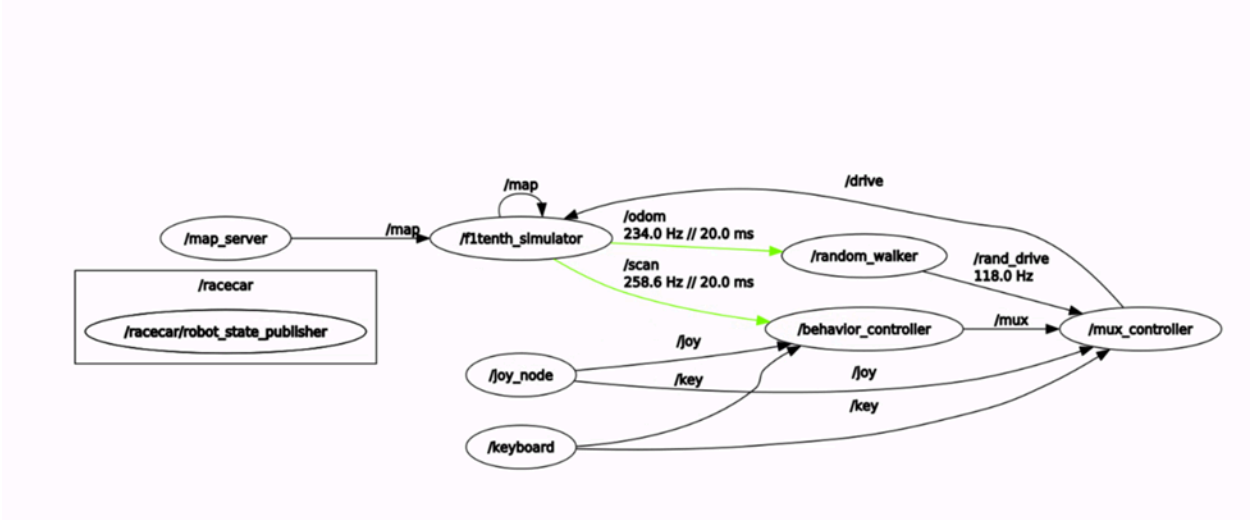
$ rosrun rqt_graph rqt_graph

Activity 11: Briefly explain the role of topics in ROS.

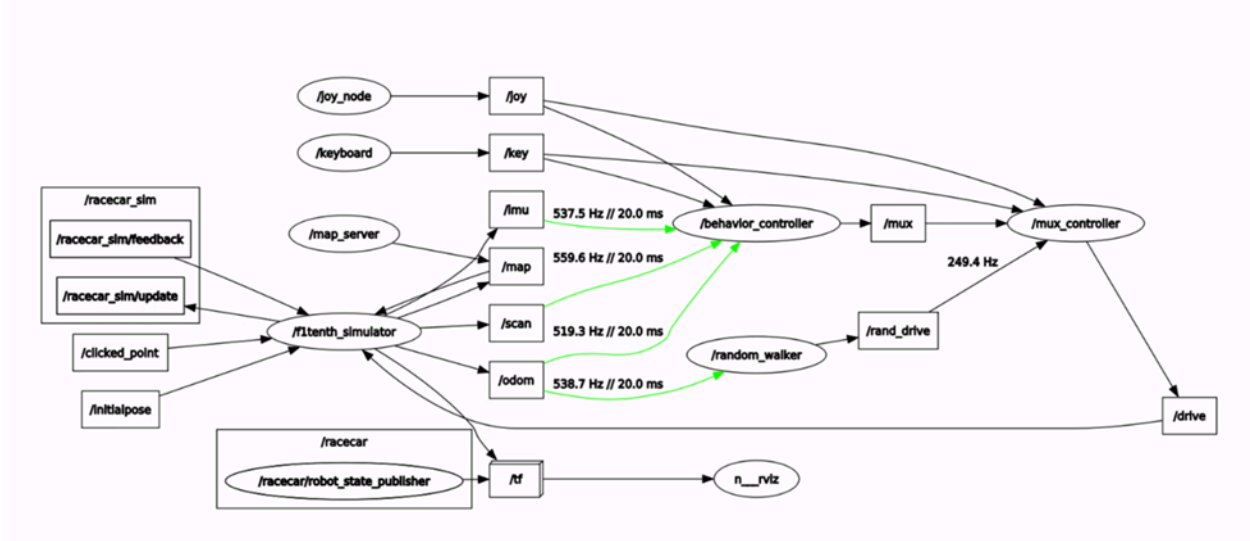
Topics are channels through which nodes in a ROS system communicate by exchanging messages. They allow nodes to communicate with each other by passing messages. Nodes can publish messages to topics and can subscribe to topics to receive messages.

Activity 12: Provide screenshots of the computation graphs in your report. What topics the node “/f1tenth_simulator” publishes to, and what topics is it subscribed to? Comment on the update rates of various topics in the computation graph and the factors that could influence the update rates.

Node only



Node / topic (all)



Topics published by "/f1tenth_simulator": /scan, /odom

Topics subscribed by "/f1tenth_simulator": /map, /racecar_sim/feedback, /racecar_sim/update, /clicked_point, /initialpose

Topic update rates

/imu: 537.5 Hz (every 20.0 ms)

/map: 559.6 Hz (every 20.0 ms)

/scan: 519.3 Hz (every 20.0 ms)

/odom: 538.7 Hz (every 20.0 ms)

There are a few factors that may affect these update rates. The physical specifications of sensors such as LIDAR or IMU determine the maximum rate at which they can provide data. Also, network speed, if the network can't process the data fast enough, the update rate may need to be reduced.

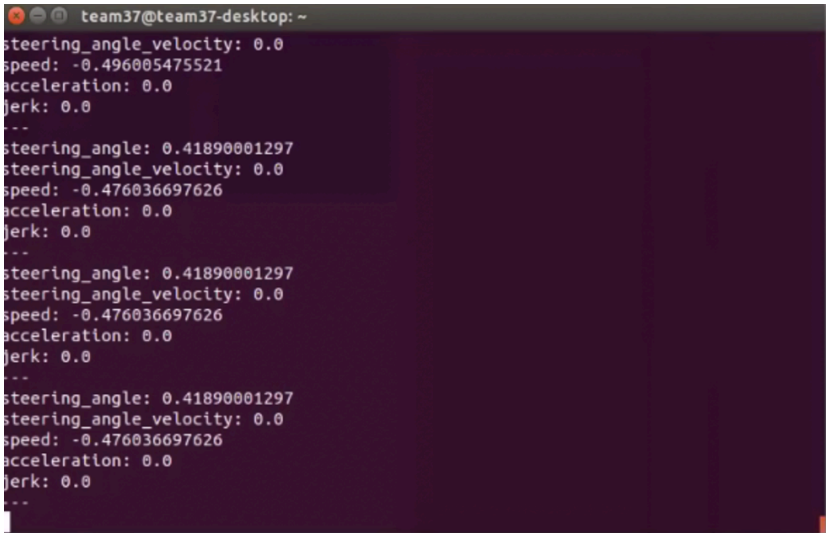
Activity 13&14:

Execute the following command to obtain information on the topic “/drive”:

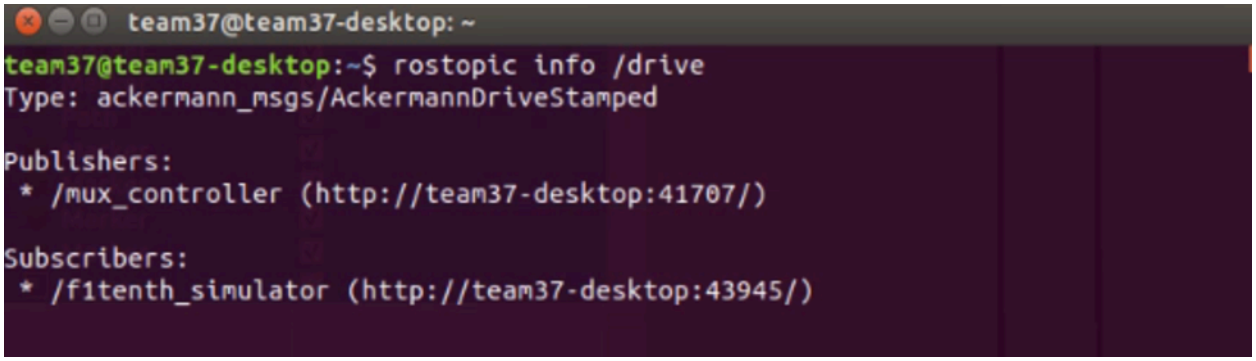
$ rostopic info /drive

The ROS command “rostopic” to echo the data in “/drive/drive”:

$ rostopic echo /drive/drive



Activity 15: Which node publishes to the “/drive” topic and at what rate? What fields of the “/drive/drive” are being updated and what do they represent?



Mux_controller node publishes to the “/drive” topic. The rate is 75 HZ.

The /drive message is likely to have several fields.Each of them represents different aspects of the drive command.

For example, velocity is one of the important fields. It typically represents the desired forward speed of the vehicle. It can be a linear velocity in meters per second.

Additionally, acceleration is also one of the important fields. It might specify the desired acceleration or deceleration rate for the vehicle, which affects how quickly it can reach the commanded velocity.

Moreover, torque is also a field that could be used instead of acceleration to control the power output of the electric motors directly.

Activity 16-17:

Use the following command to verify the data types published on topic "/map":

$ rostopic type /map

Run the command to examine the MapMetaData information published by the “map_server” on the topic “/map”:

$ rostopic echo /map/info

Activity 18: Discuss the various modes of motor control (e.g, speed, torque, etc.) and comment on their suitability for use in electrical vehicles in manual and self-driving modes.

Speed control mode is used to regulate the speed of the motor. It is typically done through a closed-loop control system that adjusts the power supply to maintain the desired speed, even with varying load conditions.

In terms of suitability for EVs, speed control is essential for cruise control systems, where the driver sets a desired speed, and the car maintains it without further pedal input in manual driving mode. In self-driving cars, speed control is a fundamental aspect which allows the vehicle to maintain, increase, or decrease speed based on traffic conditions, road signs, and navigation data.

Torque Control mode directly manages the torque output of the electric motor. In electric vehicles, torque can be changed quickly and precisely which provides fine control over acceleration and vehicle dynamics.

In terms of suitability for EVs, torque control is crucial for both manual and autonomous driving modes. For manual driving, it provides the driver with immediate response from the powertrain, which is useful for situations requiring quick acceleration or obstacle avoidance. In autonomous vehicles, torque control is used to smoothly and safely manage acceleration and deceleration which optimizes for comfort and energy efficiency.

Position Control mode is also used in standard EVs. More specifically, it is more about the precise control of the motor's rotor position and is often used in robotics and automation. It allows the motor to hold or move to a specific angular position.

In terms of suitability for EVs, position control is important in steering systems, especially in self-driving vehicles, where precise control of the steering angle is required. This might also be relevant in systems where regenerative braking is blended with friction braking which requires precise control of the motor's position for maximum energy recovery.

Activity 19 In the next few steps, we will modify the existing simulator software stack by adding and building a new node to implement the functionality of the block “Experiment”:

For this activity, we just followed the lab manual to do the following tasks.

1.First copy the file “experiment.cpp” provided on Avenue to the directory “…/ /src/f1tenth_simulator/node”. This is C++ source code for the new node.

2.Add the line “<run_depend>vesc_driver</run_depend>” to the file

“…/src/f1tenth_simulator/package.xml”.

3.Rebuild the simulator package by switching to the root catkin workspace directory and executing the “catkin_make” command. This should build the executable for the new node.

4.Add the file “experiment.launch” to the launch directory of the simulator package.

Activity 20: Open the file “experiment.cpp” and examine its content. Find the following function and explain what it does:

The "publish_driver_command" function in the ROS framework is designed to control the steering and speed of autonomous electric vehicles.

For steering control, it defines "desired_delta" to calculate the difference between the desired steering angle and the previous steering angle. This value is then clipped so that it is within the maximum allowed variation in steering angle (max_delta_servo). Finally, the resulting value is added to the previous steering angle to obtain a smooth steering angle value and stored in the last_servo variable.

For speed control, it defines "desired_rpm" to calculate speed changes. "clipped_rpm" ensures that the clipping value is within the maximum allowed variation in RPM (max_delta_rpm). The previous RPM value is then added to obtain a smoothed RPM value and the value is stored in the last_rpm variable.

Finally, the updated steering and speed commands are published using the functions "servo_pub.publish(servo_msg)" and "erpm_pub.publish(erpm_msg)" respectively.

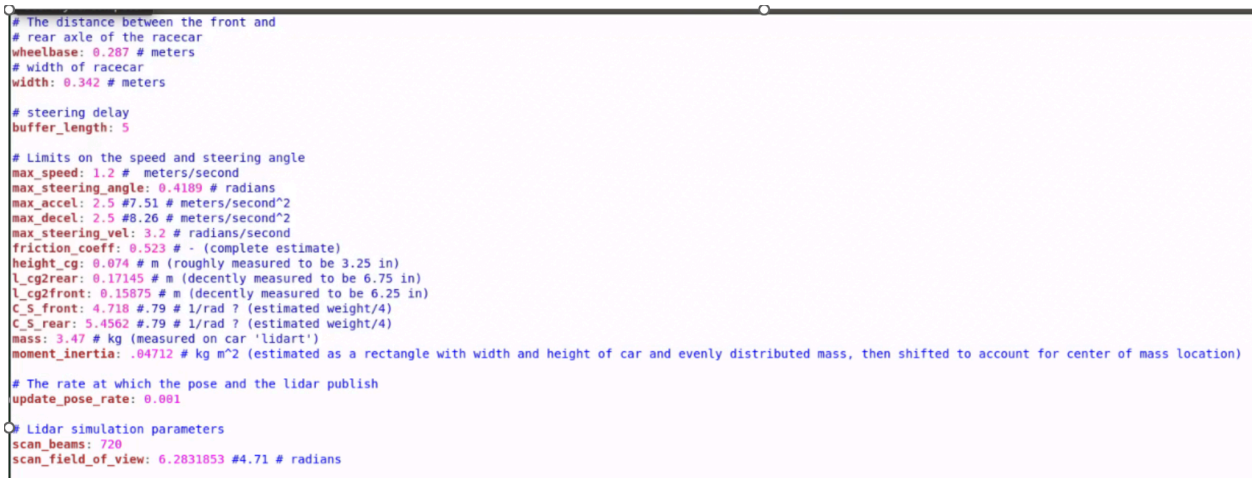
Activity 21: Find the following function and explain what it does:

The "laser_callback" function processes laser scan data from LIDAR in a ROS environment. This function calculates the midpoint of the LIDAR sensor by dividing the distance by 2. Then, it creates two vectors "msg_ranges" and "msg_intensities" to store the range and intensity values of the LIDAR sensor.

Next, the function swaps the range and intensity values for the first and second half by iterating through "msg_ranges" and "msg_intensities". This is done to correct the orientation of the lidar sensor, which may be facing forward or backward. Finally, the function creates a new ROS LaserScan message containing the range and intensity values, which includes information about the scan such as stamp, frame ID, minimum and maximum angles, angle increment, and maximum range. The corrected range and intensity values will be stored in "scan_msg" and published to the ROS topic using the litar_pub.publish() function.

Activity 22

“params.yaml” file



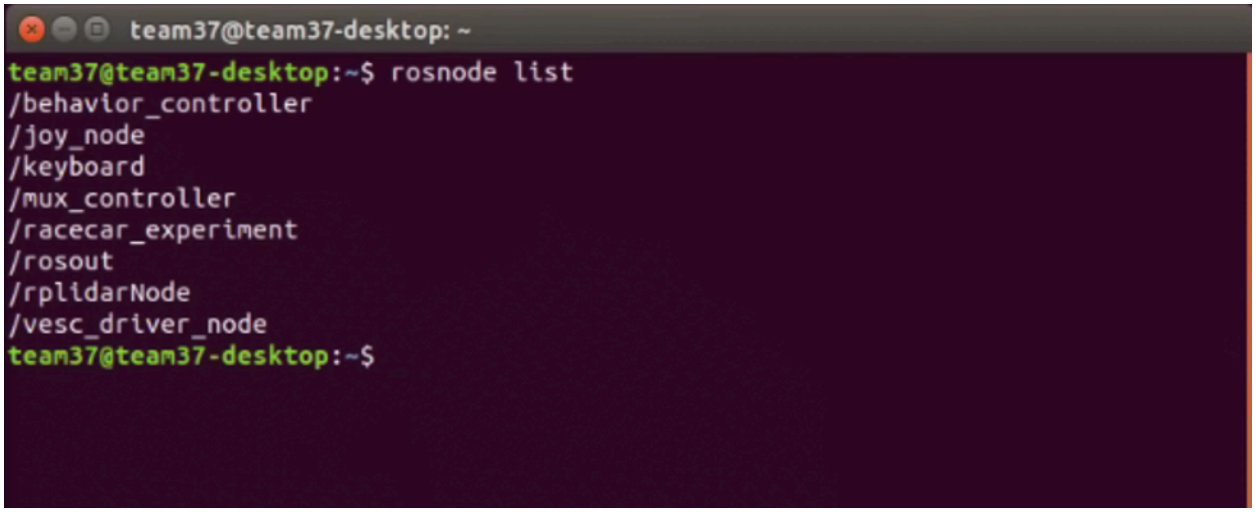
```yaml
# The distance between the front and
# rear axle of the racecar
wheelbase: 0.287 # meters
# width of racecar
width: 0.342 # meters

# steering delay
buffer_length: 5

# Limits on the speed and steering angle
max_speed: 1.2 #  meters/second
max_steering_angle: 0.4189 # radians
max_accel: 2.5 #7.51 # meters/second^2
max_decel: 2.5 #8.26 # meters/second^2
max_steering_vel: 3.2 # radians/second
friction_coeff: 0.523 # - (complete estimate)
height_cg: 0.074 # m (roughly measured to be 3.25 in)
l_cg2rear: 0.17145 # m (decently measured to be 6.75 in)
l_cg2front: 0.15875 # m (decently measured to be 6.25 in)
C_S_front: 4.718 #.79 # 1/rad ? (estimated weight/4)
C_S_rear: 5.4562 #.79 # 1/rad ? (estimated weight/4)
mass: 3.47 # kg (measured on car 'lidart')
moment_inertia: .04712 # kg m^2 (estimated as a rectangle with width and height of car and evenly distributed mass, then shifted to account for center of mass location)

# The rate at which the pose and the lidar publish
update_pose_rate: 0.001

# Lidar simulation parameters
scan_beams: 720
scan_field_of_view: 6.2831853 #4.71 # radians
```

Activity 23&24

We ran the command “$ roslaunch f1tenth_simulator experiment.launch” to start operating the vehicle.

We also executed the command “$ rosnode list” to find the running nodes in the system.

Activity 25: Report the list of running nodes and briefly explain their roles.



/behavior_controller

This node controls the behavior of the autonomous electric vehicles.

/joy_node

This node is responsible for connecting the joystick

/keyboard

This node is responsible for processing keyboard input and converting it into ROS messages.

/mux_controller

This node is a multiplexer that controls the movement or operation of the vehicle.

/racecar_experiment

This node is created for vehicles experimental racing.

/rosout

This is the standard node in the ROS system, which is used to record messages from other nodes

/rplidarNode

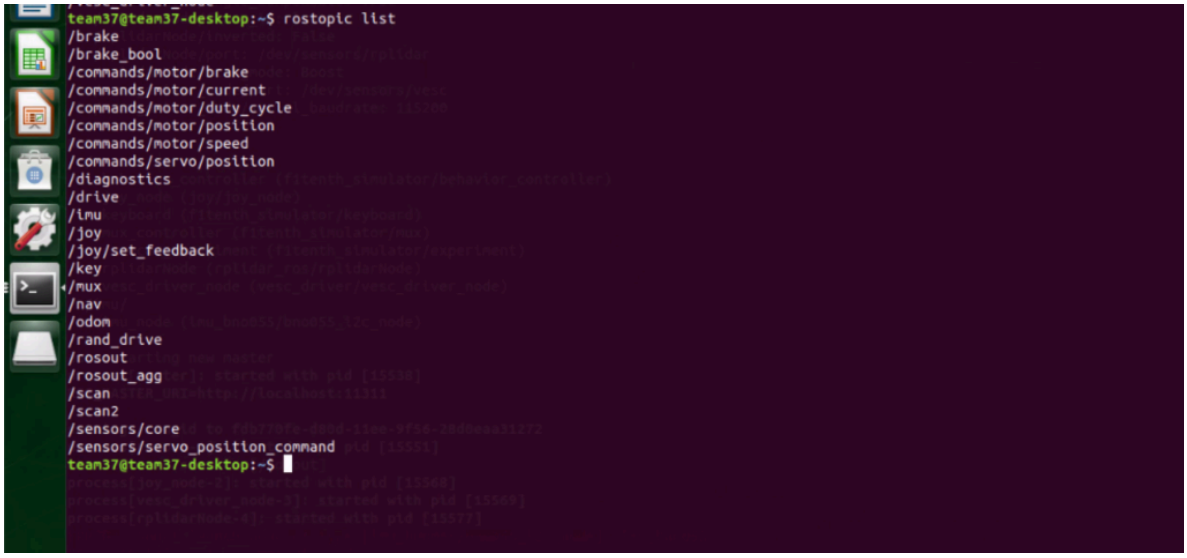
This node is responsible for the connection to RPLIDAR

/vesc_driver_node

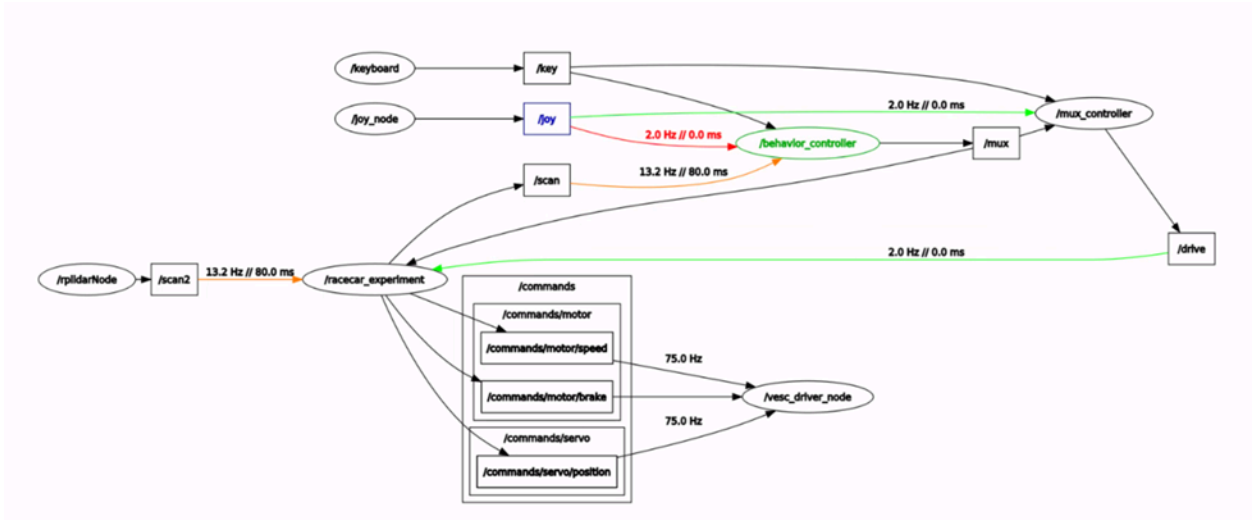
This node communicates with the VESC and manages the control commands for the motors.

Activity 26:List the topics published over the ROS environment by executing the command:

$ rostopic list



Activity 27: Report the ROS topics associated with the VESC ROS driver. Examine the rate at which the motor speed and the servo steering angle commands are published to their respective topics. What is the rate of execution for the VSEC driver node "/vesc_driver_node"? What factors affect these rates?

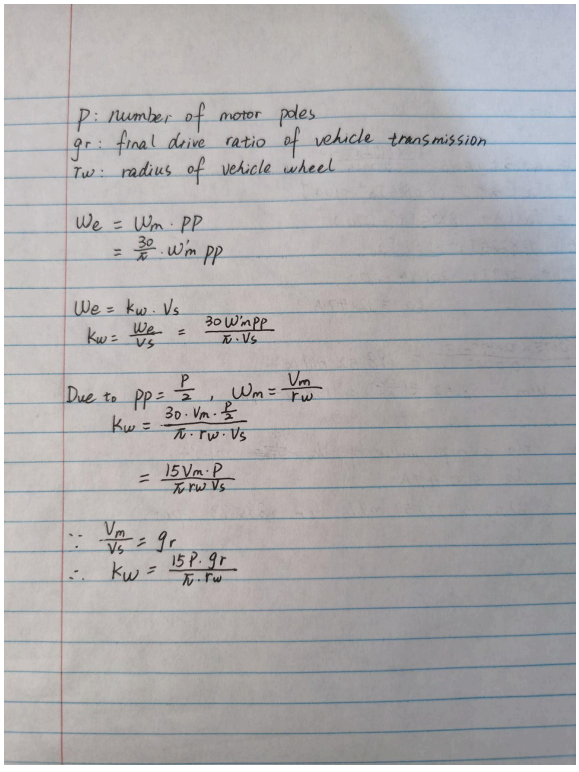


The execution frequency of the VESC driver node, designated as "/vesc_driver_node", is set at 75 Hz. This frequency correlates directly with the invocation rate of the publish_driver_command function, defined by the driver_smoother_rate. The chosen rate is a balance between various factors such as the processing power of the onboard hardware, the computational intensity of the control algorithms, and the operational requirements for vehicle performance. Opting for a higher command publishing frequency typically enhances the vehicle's responsiveness and ensures a more refined control experience. However, it also imposes more rigorous processing demands on both the hardware and the control software.

Activity 28

For this activity, we just followed the lab manual to do the required tasks. Press and release the LB on the Joystick once to enable manual driving. Use the two mini-sticks to send speed and steering commands to the vehicle. If the vehicle is responding as expected, you can place it on the ground and start driving it around. Make sure that you stay close enough to the vehicle so you would not lose communication between the F710 Joystick and Jetson Nano. After finishing your drive, shut down the program by pressing ctrl-c in the command window.

Activity 29: Derive the given formula for $k\omega$.



Activity 30:

According to our testing, the offset value $\delta 0$ is 0.445. This value will simply drive the vehicle in a straight line without giving any steering commands.

```
# Ackermann to VESC parameters

speed_to_erpm_gain: 3182.18
speed_to_erpm_offset: 0.0
steering_angle_to_servo_gain: -0.88
steering_angle_to_servo_offset: 0.445000

driver_smoother_rate: 75.0 # messages/sec
```

Activity 31:

According to our measurements, the distance between the start and end points 2r =133cm, vehicle wheelbase L=30.5cm.

Activity 32: Use the results of a calibration experiment to recalculate the value of k_δ. Update the value of this parameter in "params.yaml".

δ = $\tan^{-1}$(l/r)

= 24.64

δ_d = 0.4189

$\overline{K}$ = -0.88

K_δ = $\delta_d/\delta^*\overline{K}$

K_δ = (0.4189)/(24.64*-0.88)

= -0.0193

Activity 33: Examine the following line from the file "experiment.cpp" and explain what it does:

update_command =
n.createTimer(ros::Duration(1.0/driver_smoother_rate),
&RacecarExperiment::publish_driver_command, this);

The update_command is a timer that schedules callbacks at the specified rate of 1.0/driver_smoother_rate. Here, the scheduled callback is the publish_driver_command function, which calculates the drive data: smoothed_servo and smoothed_rpm values, and then publishes this data to the VESC.

Activity 34: Execute the following command:

$ rostopic hz /commands/motor/speed

The above command displays the publishing rate of the speed topic.

Activity 35: Briefly explain how some use cases for the following ROS commands based on the activities of this lab:

$ rosrun : The rosrun command allows us to execute a file from a package without having to specify the whole path to the package it is contained in. For example, $**rosrun rqt_graph rqt_graph**, runs the rqt_graph plug-in in the rqt_graph package without having to specify what the path is of that package.

$ roslaunch : The roslaunch command allows us to launch multiple nodes at once, and uses an XML file as an input which contains the relevant nodes to be launched. For example, **$roslaunch f1tenth_simulator experiment.launch**, launches several nodes contained in the file experiment.launch, which we are able to view using the command rosnode list.

$ rosnode : Allows us to view info about the existing nodes in the ROS environment through commands such as info, list, etc. For example, **$rosnode list,** will display a list of existing nodes in the ros environment.

$ rostopic : Allows us to view info about the existing topics in the ROS environment through commands such as echo, info, etc. For example, **$rostopic info /drive**, shows us the drive topic’s subscribers and publishers.