1. **What do those parameters represent? Why are those parameters essential to control the motor?**

   The parameters are the settings that control the behaviour of the motor, particularly its rotation direction and speed. These parameters are essential because they determine how the motor responds to commands and external factors. For instance, inverting the motor rotation direction ensures that the motor operates in sync with the vehicle's forward motion. Setting PID parameters for speed control enables precise adjustment of the motor's speed based on inputs and feedback, enhancing overall performance and efficiency.

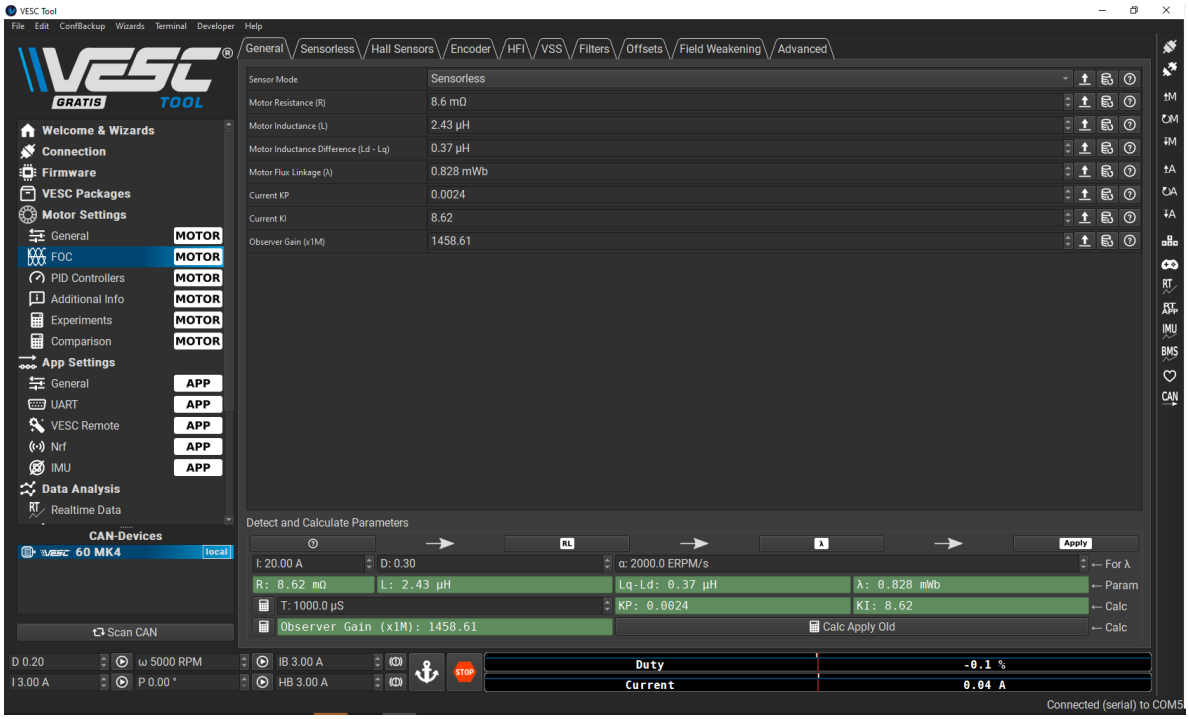


Figure 1: VESC Tool Parameter Identification

2. **Explain the importance of PID speed control parameters in relation to the operation of McMaster AEV?**

   Accurately regulating the motor's speed is essential for the McMaster AEV's operation, and the PID speed control parameters play a crucial role in achieving this. These parameters determine how the motor responds to changes in speed commands and external factors such as load variations. Properly tuning the PID parameters is essential as it ensures smooth acceleration, deceleration, and steady-state speed, thereby improving the AEV's overall performance, stability, and efficiency.

3. **First apply speed command to VESC and record how the speed, current and voltage change in time. Please change the speed gradually to a maximum speed command of 7500 rpm. (Recommended: start with a speed of 2500, then 5000 and finally 7500**

rpm, Take a screenshot of Your RPM and FOC graphs. Log your data for each speed point and plot the graphs for RPM, FOC current (d- and q-axis currents), FOC voltage (d- and q-axis voltages).

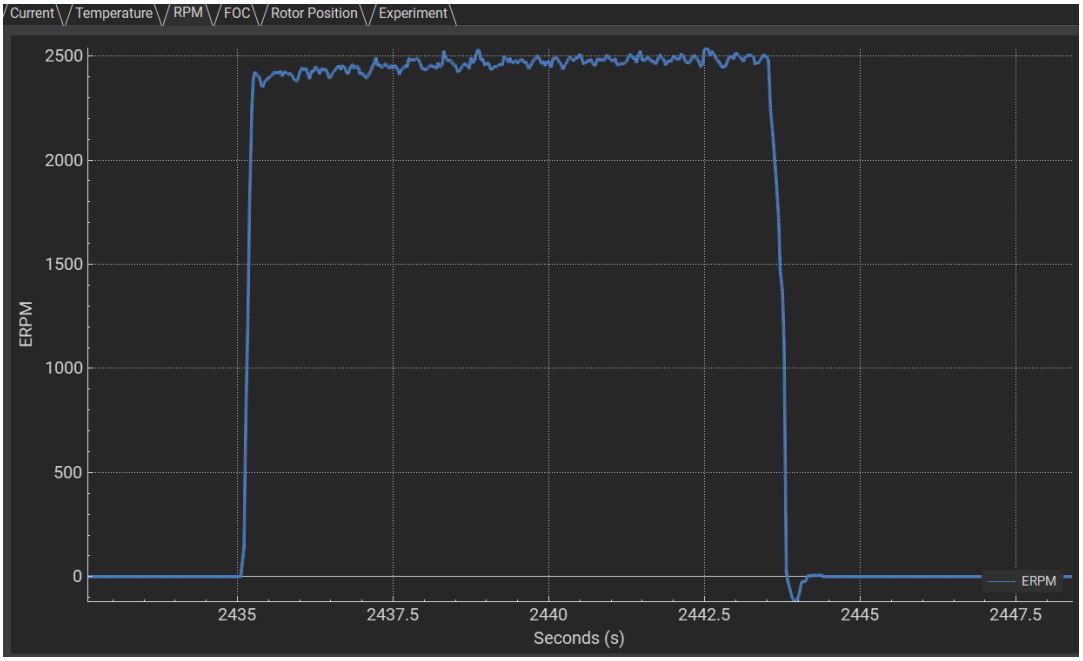
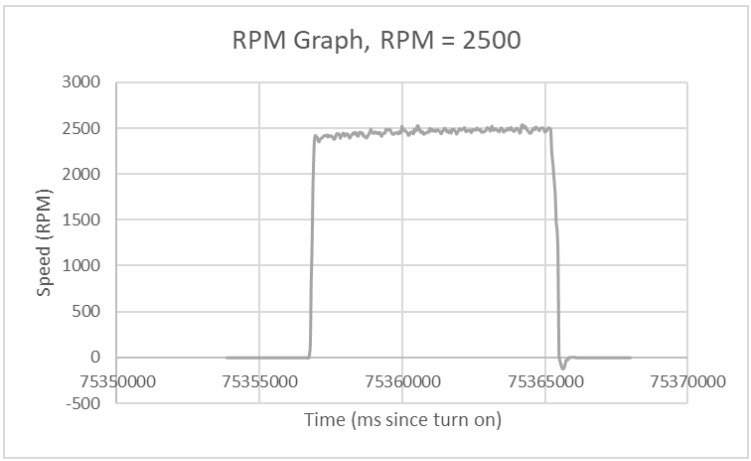
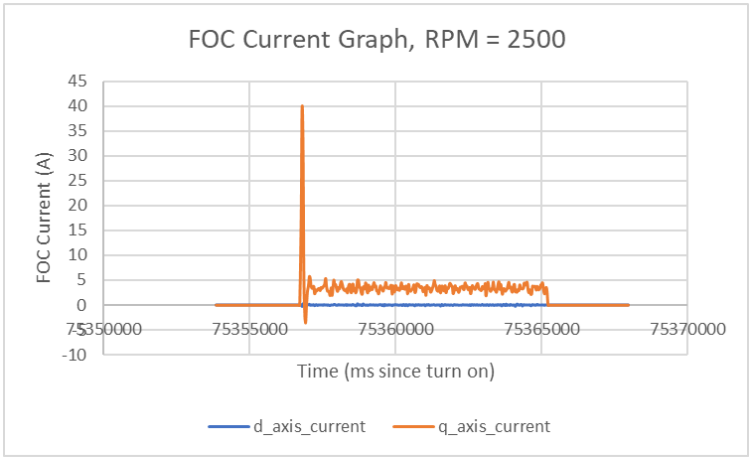


Figure 2: EPRM Graph for set speed of 2500 rpm

Figure 3: FOC Graph for set speed of 2500 rpm





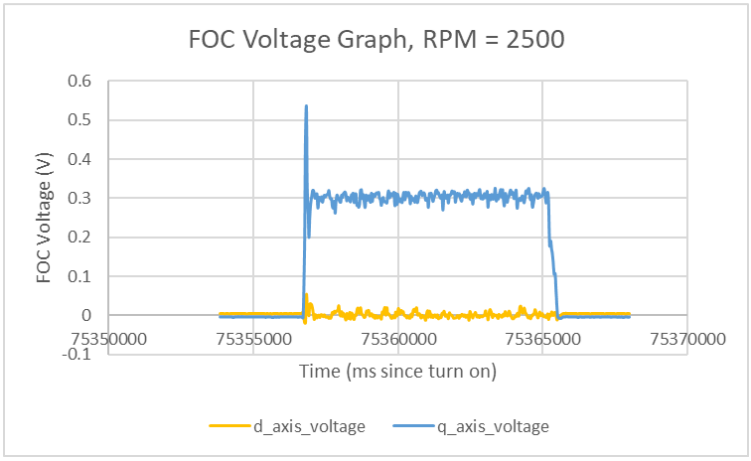


Figure 4: Plotted Graphs for 2500 rpm

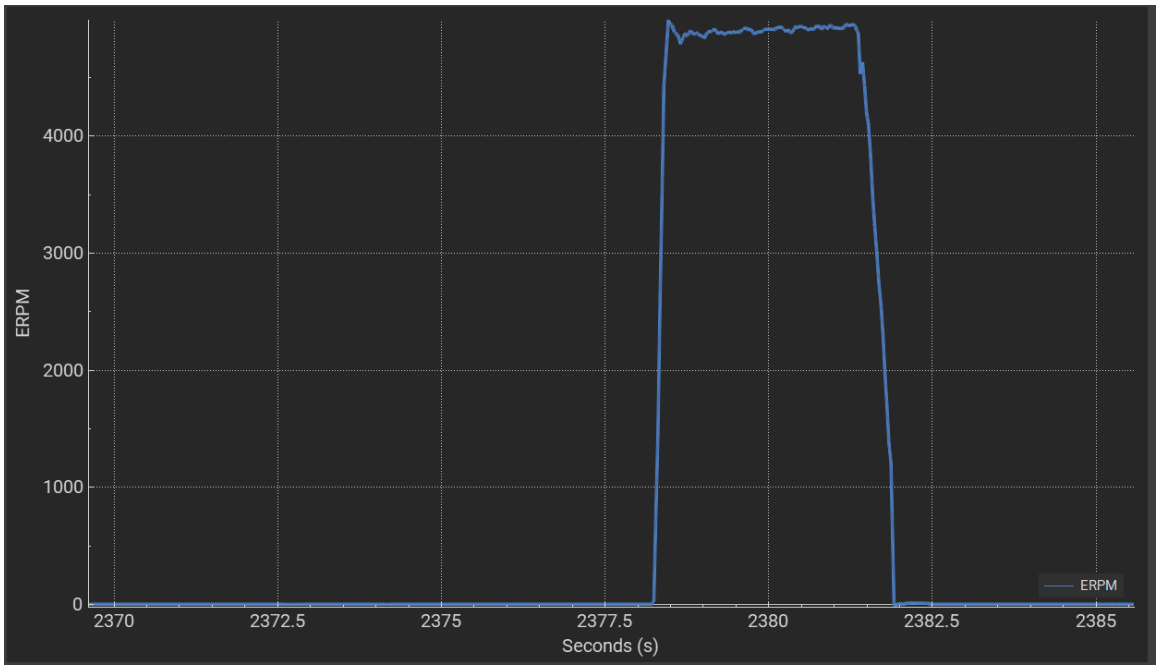


Figure 5: EPRM Graph for set speed of 5000 rpm

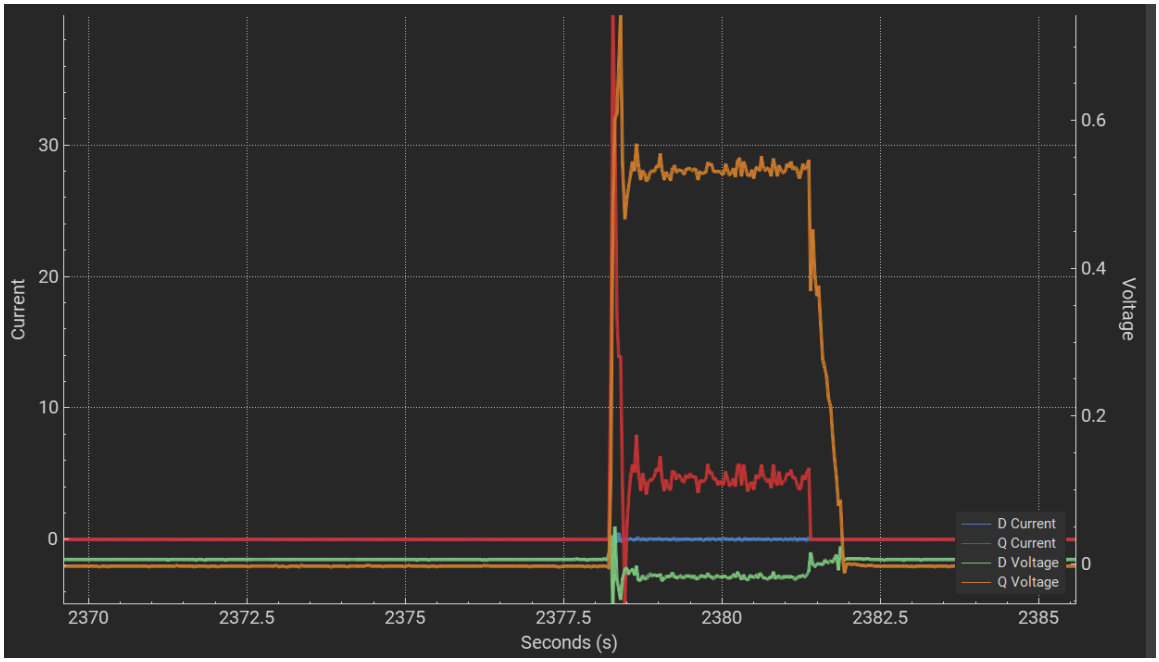
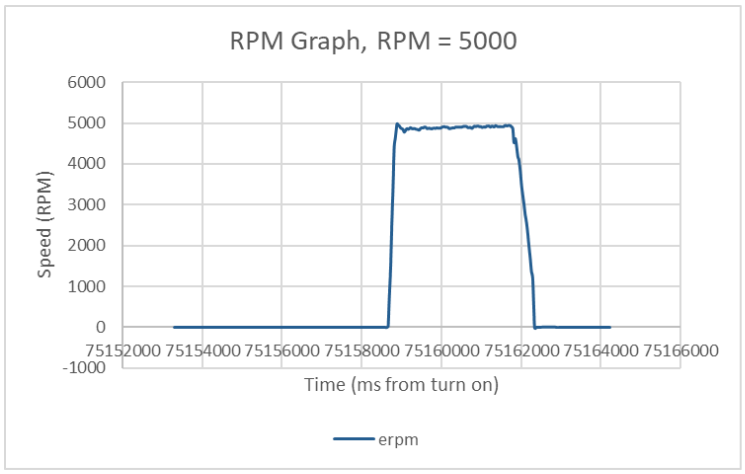
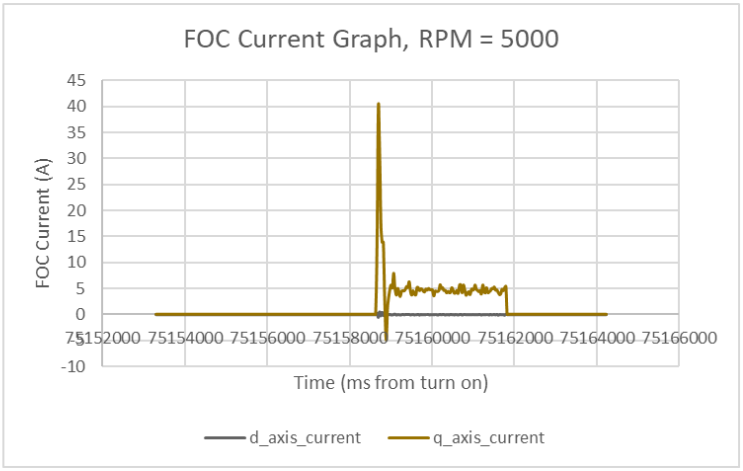


Figure 6: FOC Graph for set speed of 5000 rpm





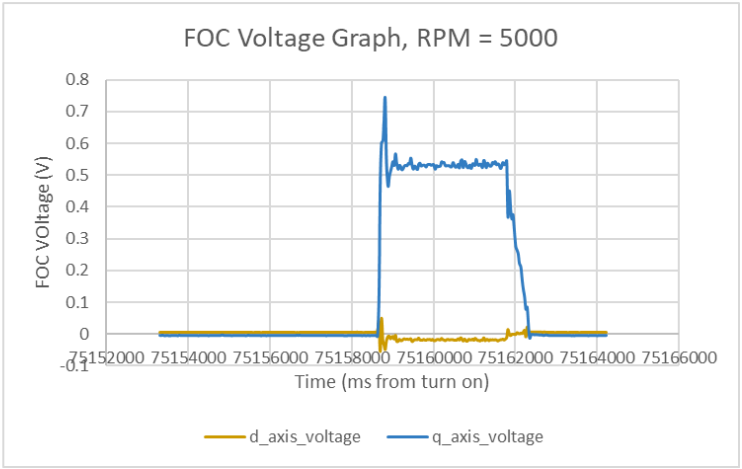


Figure 7: Plotted Graphs for 5000 rpm

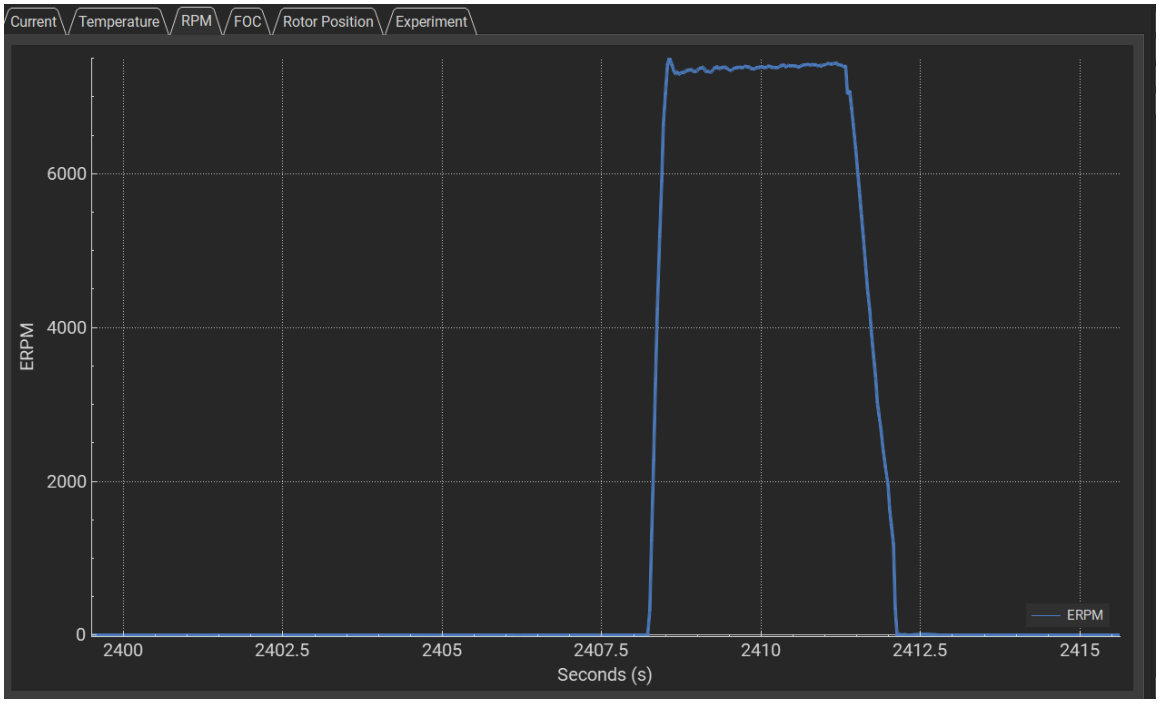


Figure 8: EPRM Graph for set speed of 7500 rpm

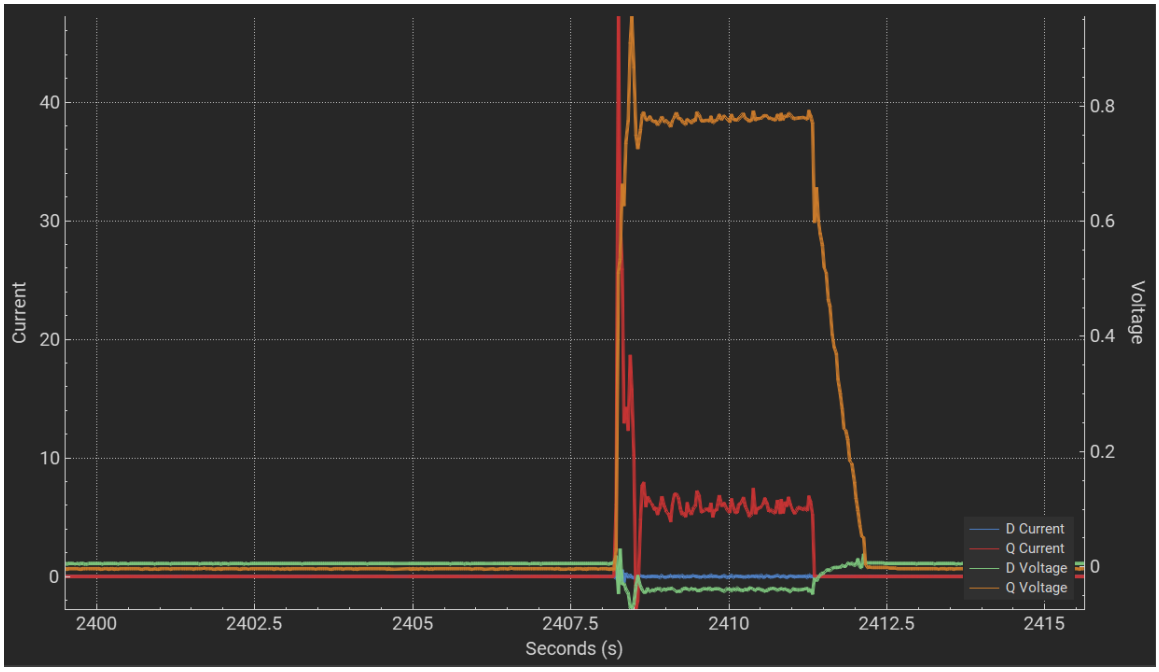
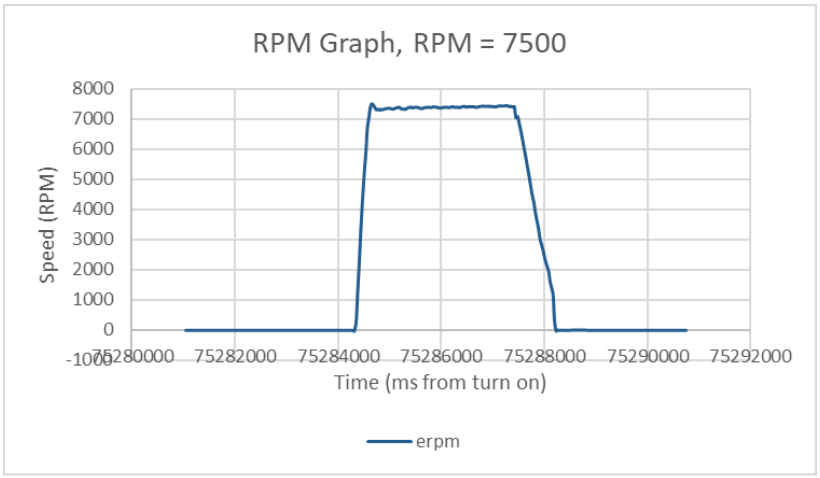
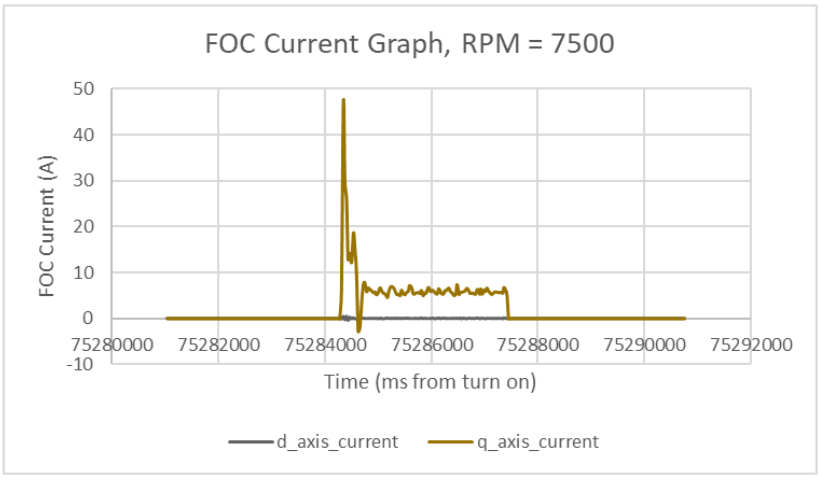


Figure 9: FOC Graph for set speed of 7500 rpm





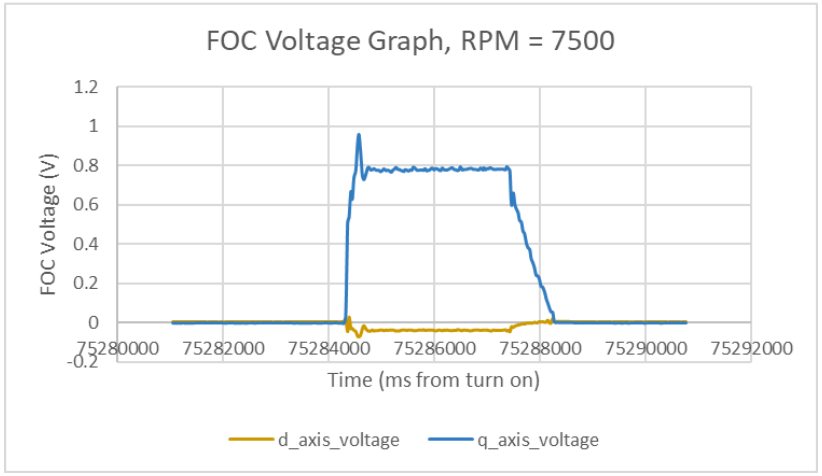


Figure 10: Plotted Graphs for 7500 rpm

4. **Next test the motor operation with duty cycle input. Please maintain the duty cycle lower than 20%. (Recommended: start with a duty cycle of 0.10, then 0.15 and finally 0.2, Take a screenshot of Your RPM and FOC graphs. Log your data for each speed point and plot the graphs for RPM, FOC current (d- and q-axis currents), FOC voltage (d- and q-axis voltages). In this experiment, do not apply current command.**

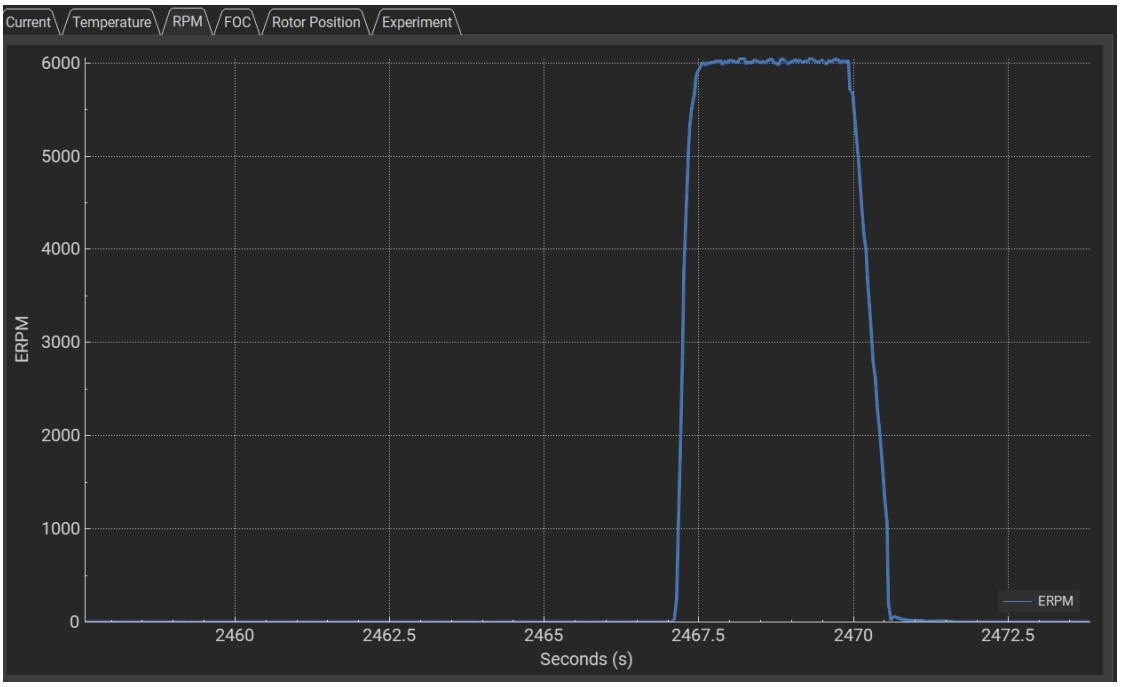


Figure 11: EPRM Graph for set duty cycle of 10% (0.1)

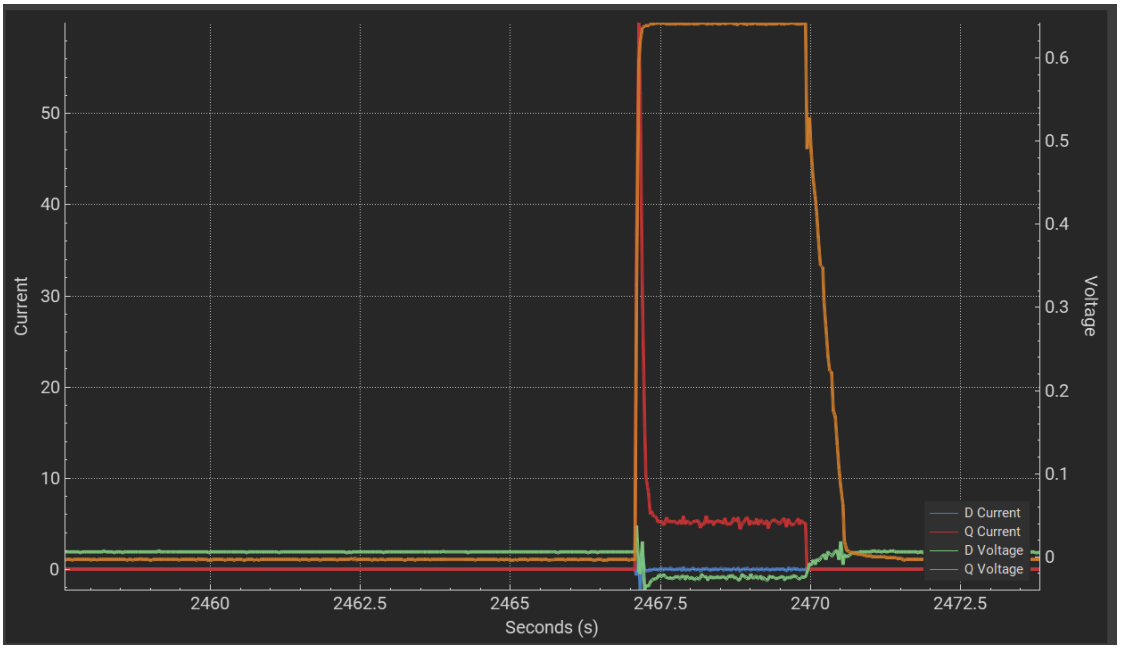
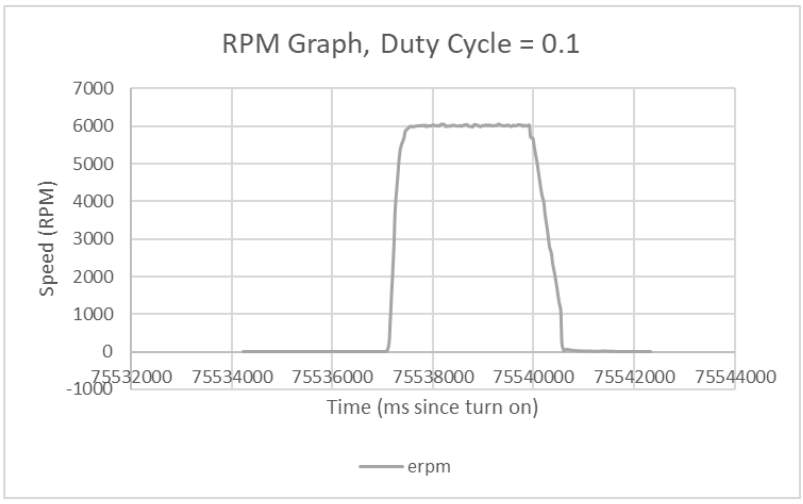
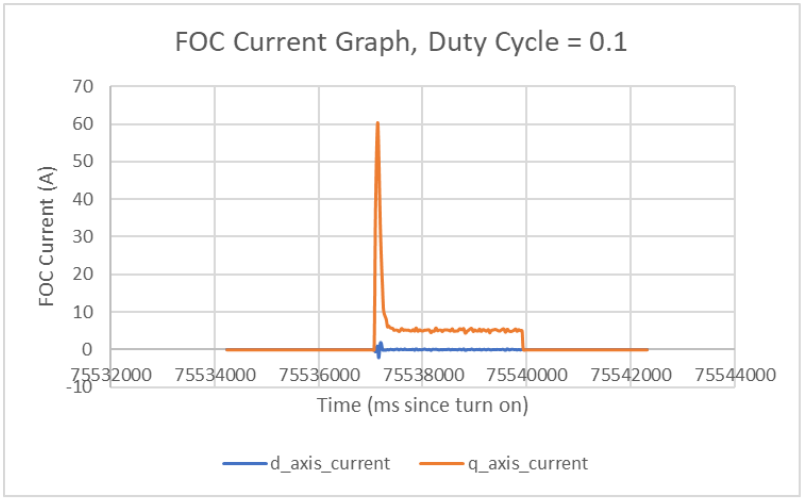


Figure 12: FOC Graph for set duty cycle of 10% (0.1)





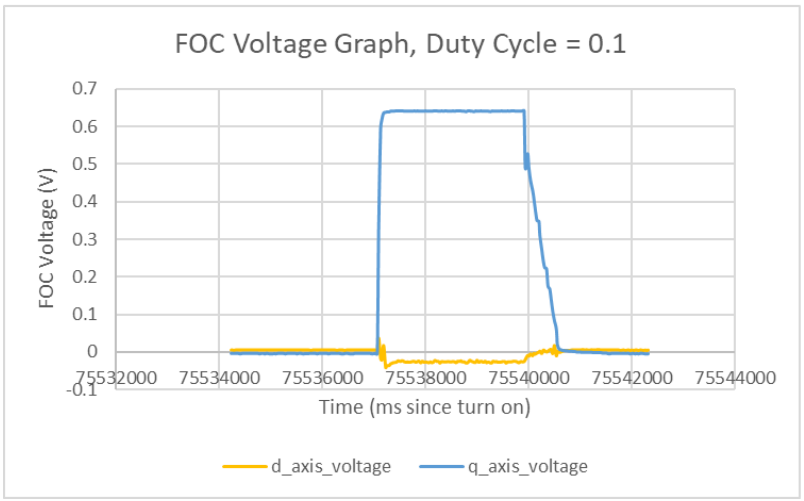


Figure 13: Plotted Graphs for 0.1 duty cycle

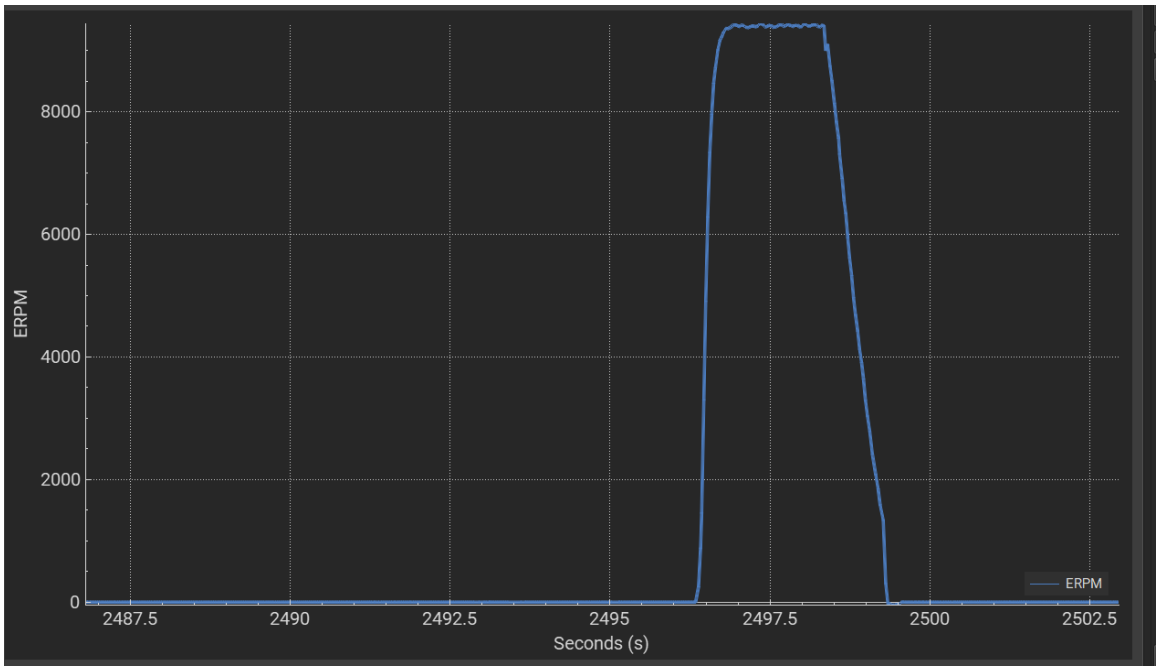


Figure 14: EPRM Graph for set duty cycle of 15% (0.15)

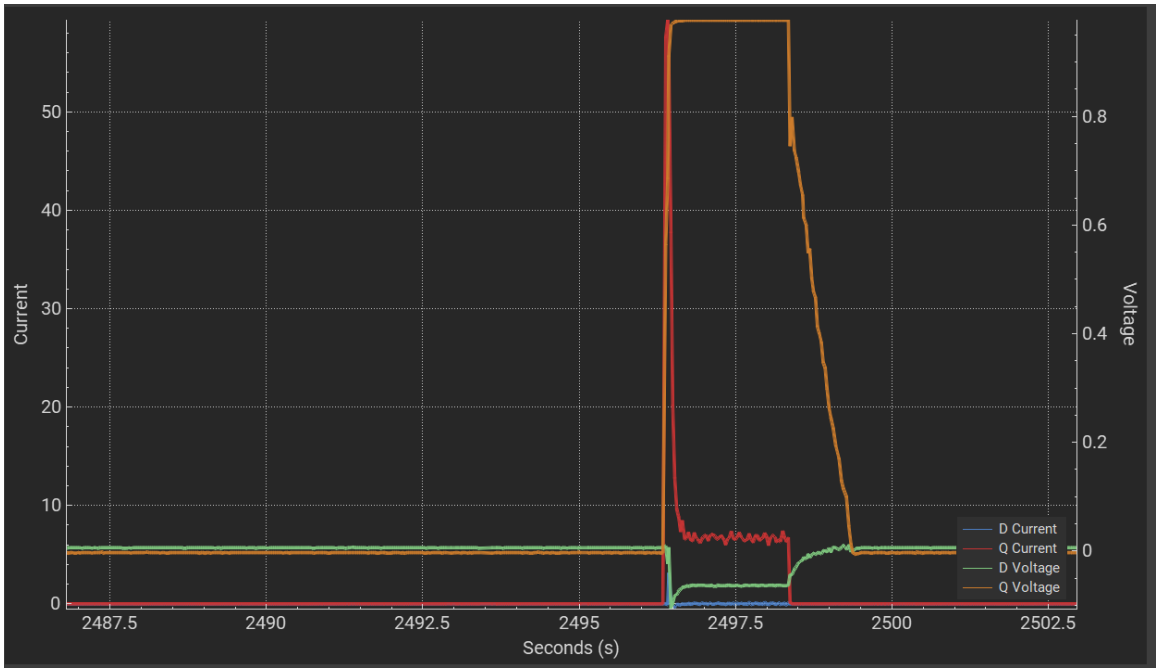
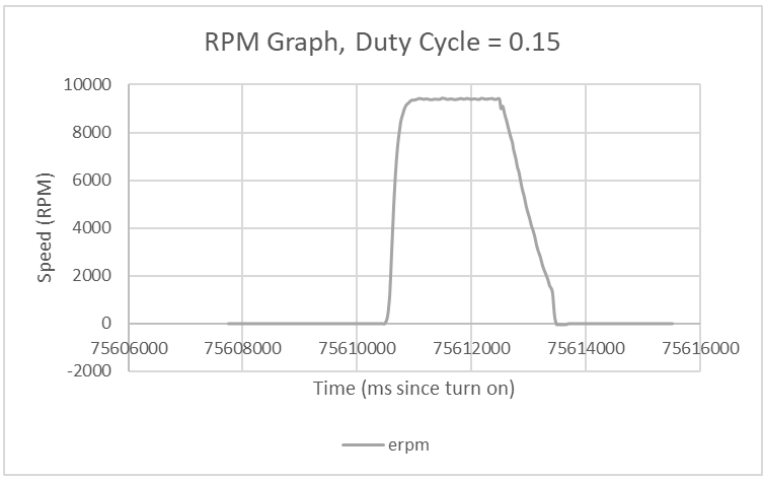
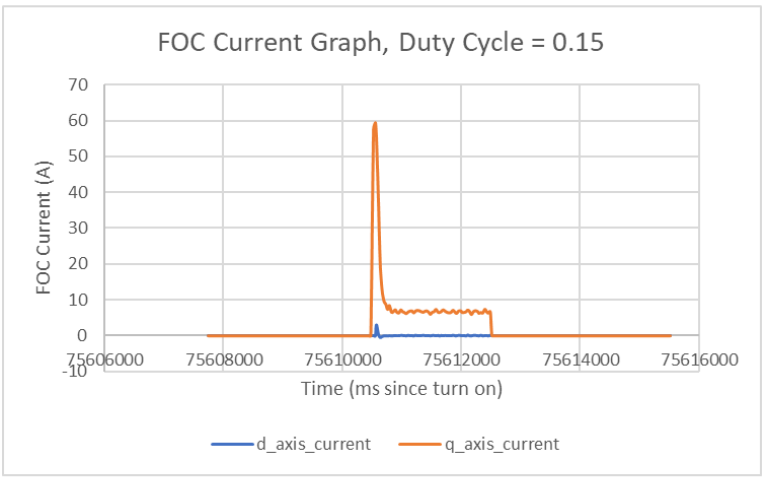


Figure 15: FOC Graph for set duty cycle of 15% (0.15)





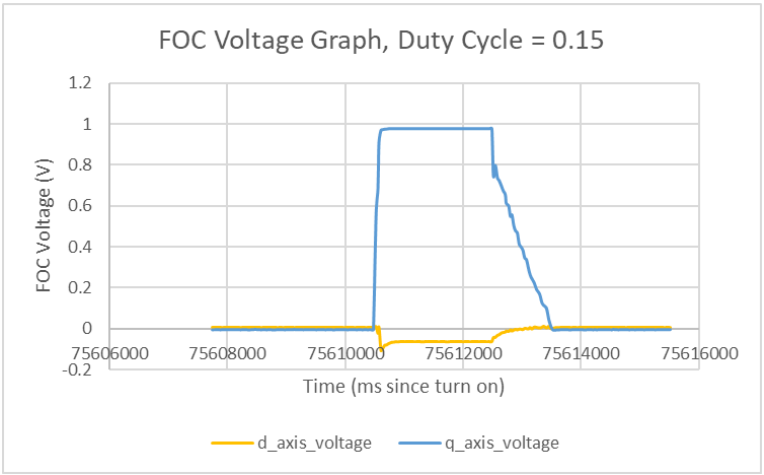


Figure 16: Plotted Graphs for 0.15 duty cycle

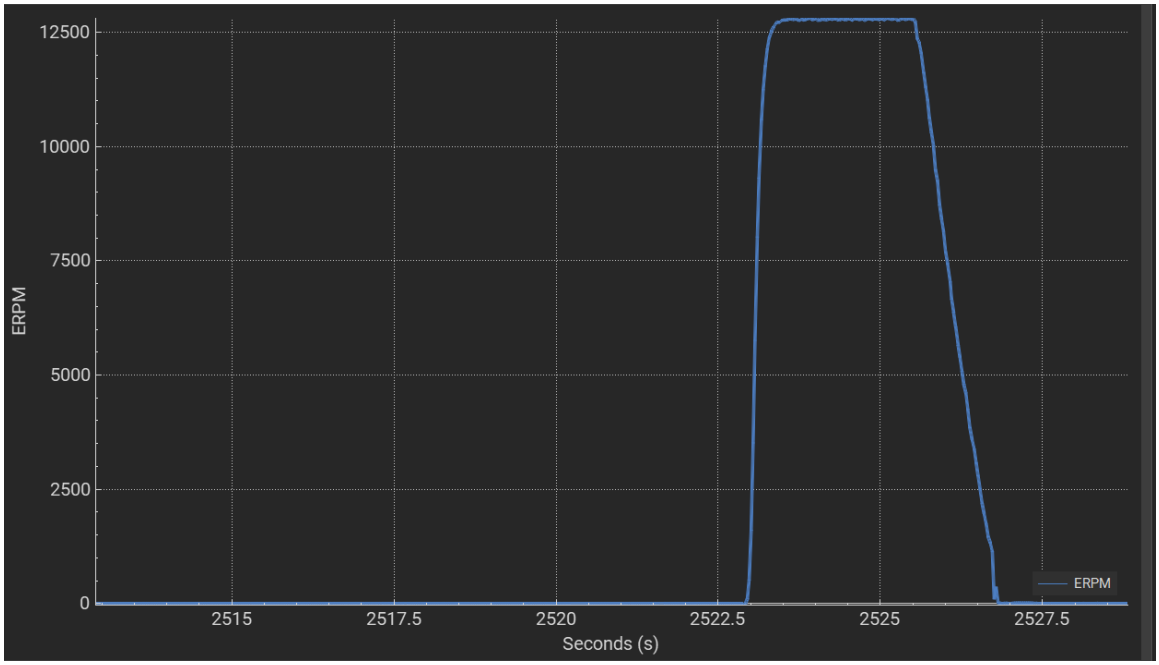


Figure 17: EPRM Graph for set duty cycle of 20% (0.2)

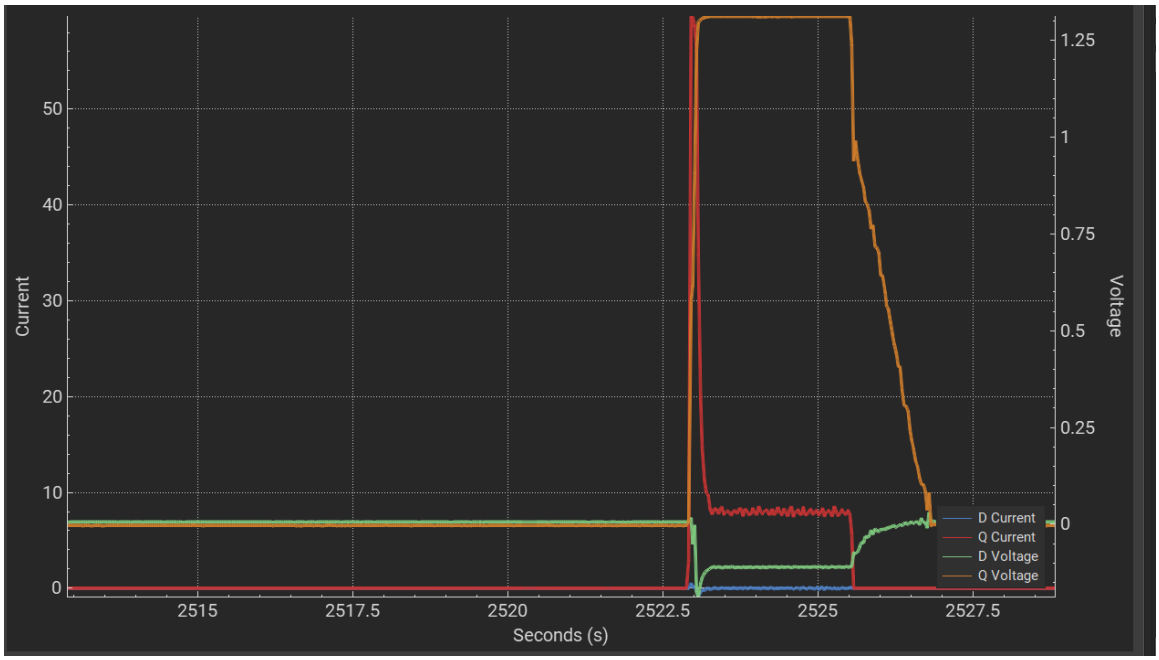
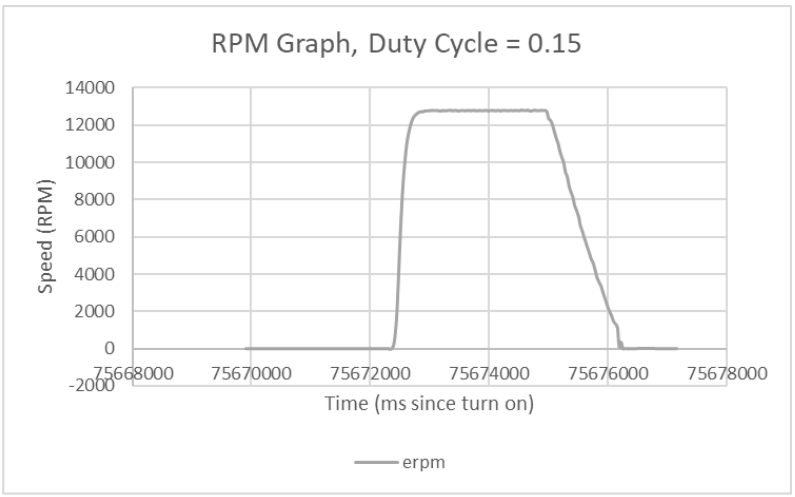
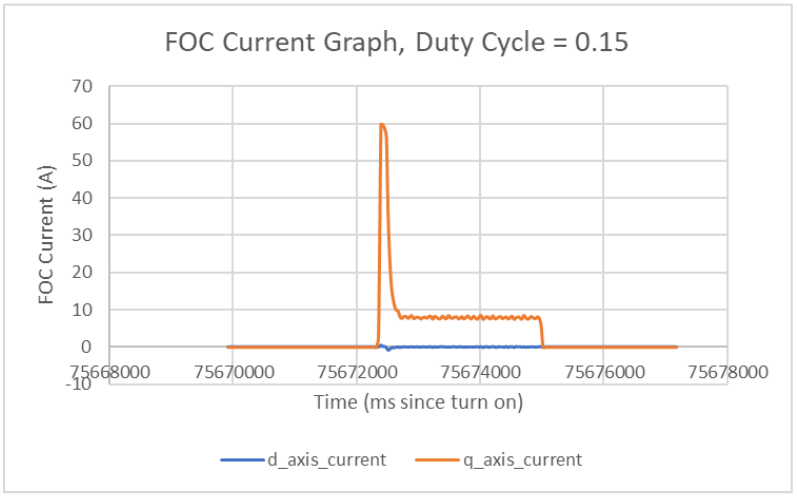


Figure 18: FOC Graph for set duty cycle of 20% (0.2)





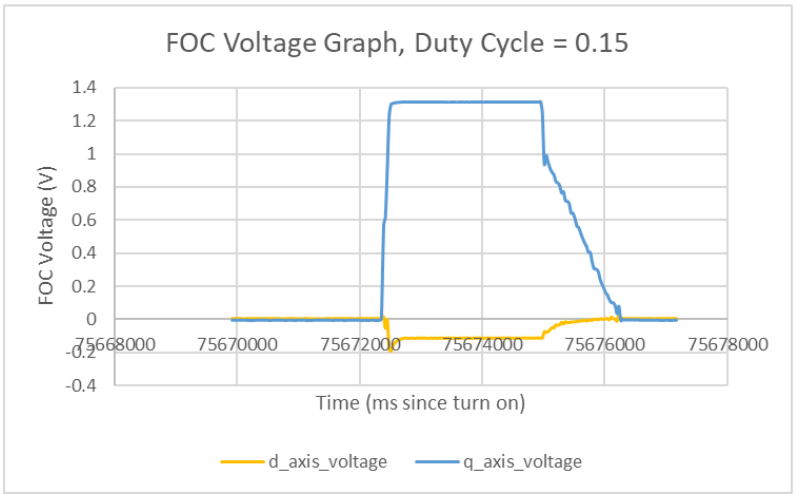


Figure 19: Plotted Graphs for 0.2 duty cycle

5. **What difference do you observe between the operations with the speed command and duty cycle command? Explain the difference using the real-time voltage, current, and speed waveforms.**

   There are two types of commands for motor operations - speed command and duty cycle command. The main difference is how the motor responds to the input signals. Speed command controls the motor's rotational speed directly, while duty cycle command modulates the average voltage supplied. To understand the difference in control behaviour and motor performance between these two commands, real-time voltage, current, and speed waveforms provide visual feedback.

6. **What is the definition of the d and q-axis currents? Why is the d-axis current always zero? Please check your lecture notes.**

   The d and q-axis currents are parts of the current that flow through the motor windings, represented in a coordinate system that aligns with the motor's magnetic field. The d-axis current is responsible for controlling the motor's torque, while the q-axis current is responsible for controlling the motor's magnetic flux. In a synchronous motor, the d-axis current is always zero when the rotor and stator magnetic fields are aligned, which results in maximum torque production.

7. **Record the experiment results in a .csv file and plot them for your report. Explore how you can log the experiment with Realtime Data feature in the VESC Tool. Attach a screenshot of your plot tab in your report. Attach two graphs from your .csv file: one for RPM and the other one for q-axis current.**

   By logging data during the experiment and plotting RPM and q-axis current graphs, we can analyze the motor's response to different speed commands and further understand its behaviour under varying conditions.

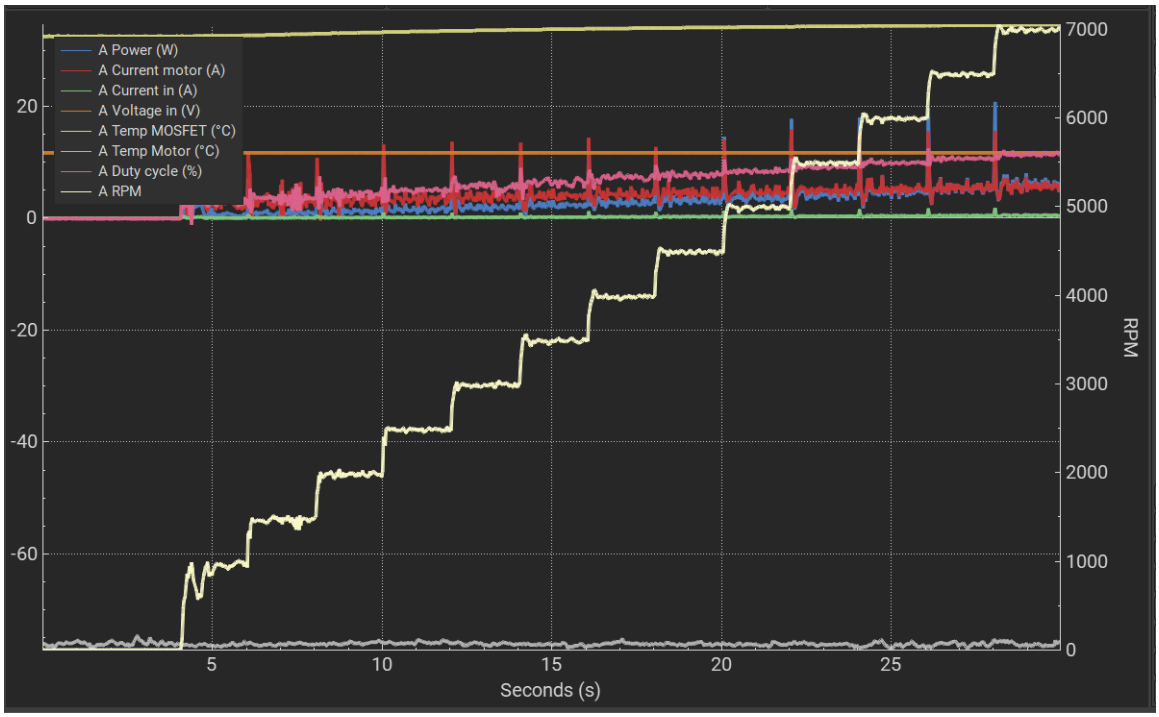


Figure 20: Plot tab for Question 7

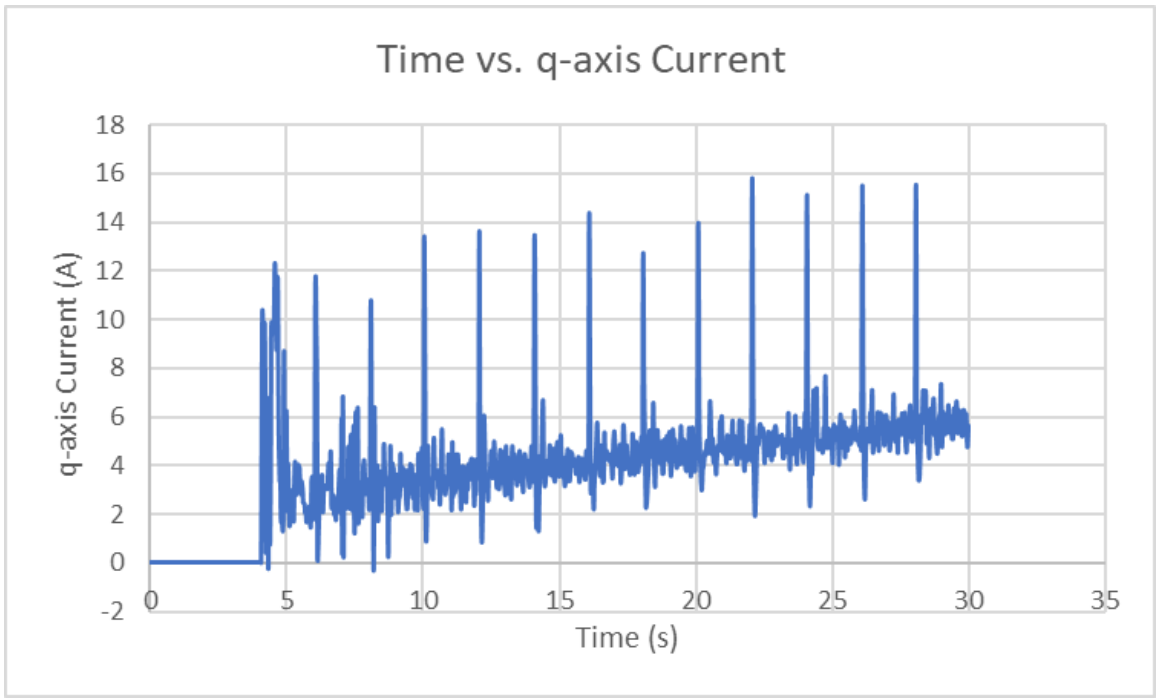


Figure 21: Time (s) vs. Q-axis Current (A)

1. **What is the role of q-axis current in the operation of a permanent magnet synchronous motor? What is the effect of q-axis current on torque generation?**

The q-axis current in a permanent magnet synchronous motor is important for generating torque by interacting with the rotor's permanent magnets. Controlling the magnitude and phase angle of the q-axis current allows for precise control of motor torque, allowing efficient performance control in various operating conditions.

2. **What is the difference between installing the VESC package directly with apt-get and cloning from GitHub?**

Installing the VESC package directly with apt-get provides a stable and easy-to-install version from your the package repository, while cloning from GitHub gives you access to the latest version of the package with more customization but requires extra steps to install and may even have some issues with stability.

3. **Explain what the static_cast in the code is used for. Why is it necessary to cast these values?**

In the code, static_cast is used to convert data types of certain variables. All four of the statis_cast lines cast the value pointed by 'payload_.first+ ##' to an unsigned 32-bit integer then shift it by a certain amount of bits. These casts are necessary to make sure that the values obtained from the payload are seen as 32-bit unsigned integers so that operations like shifting to the left can be performed without any errors.

4. **Describe what each line of the added code block in Figure 1 does? How does it contribute to parsing the q-axis current data? Why do you think the bytes are shifted (<<) by 24, 16, and 8 bit respectively? What does this accomplish in the content of data parsing?**

(static_cast<uint32_t>(*(payload_.first + 17)) << 24)

This line retrieves the value at the memory location pointed to by 'payload_.first + 17', then it casts it to an unsigned 32-bit integer 'uint32_t', and then shifts it left by 24 bits. Shifting left by 24 bits aligns this byte to the most significant byte position in a 32-bit integer.

(static_cast<uint32_t>(*(payload_.first + 18)) << 16)

Similar to the previous line, this retrieves the value at 'payload_.first + 18', casts it to 'uint32_t', and shifts it left by 16 bits. Shifting left by 16 bits aligns this byte to the next most significant byte position in the 32-bit integer.

(static_cast<uint32_t>(*(payload_.first + 19)) << 8)

Again, this retrieves the value at 'payload_.first + 19', casts it to 'uint32_t', and shifts it left by 8 bits. Shifting left by 8 bits aligns this byte to the next most significant byte position in the 32-bit integer.

static_cast<uint32_t>(*(payload_.first + 20))

This retrieves the value at ‘payload_.first + 20’ and casts it to ‘uint32_t’. It doesn't shift this value since it's already in the least significant position.

By shifting each byte to its appropriate position in a 32-bit integer and combining them, the code reconstructs a 32-bit integer value from four consecutive bytes in memory. The shifting also ensures that each byte contributes its correct portion to the final value, effectively reconstructing the original q-axis current data.

5. **Explain the purpose of declaring the current_qaxis()method in the vesc_packet.h file. How does this relate to the changes made in the .cpp file? How does the declaration enable the VESC driver to publish this data?**

   The purpose of declaring the current_qaxis() method in the vesc_packet.h file is to define a function prototype that will be used by the VESC driver to access the q-axis current data from the motor's controller. This method is then implemented in the .cpp file, where the actual code to retrieve the q-axis current from the VESC's data packet is written. The declaration in the .h file tells the compiler what the function looks like, while the implementation in the .cpp file defines what the function does. By declaring and implementing this function, the VESC driver can call current_qaxis() to obtain the q-axis current data and publish it, which is important for applications that need to monitor or control the motor's torque.

6. **What is the purpose of adding state_msg->state.current_q = values->current_qaxis(); to the vesc_driver.cpp file?**

   The purpose of adding state_msg->state.current_q = values->current_qaxis(); to the vesc_driver.cpp file is to assign the q-axis current value retrieved by the current_qaxis() method to the current_q field of the state message. This line of code populates the current_q field with the actual current data so that when the state message is published, it includes the latest q-axis current information. This allows any nodes subscribing to the VESC state topic to receive up-to-date torque-related data from the motor.

7. **How does the addition of current_q to the VescStateStamped message affect the information that the VESC node publishes?**

   The addition of current_q to the VescStateStamped message affects the information that the VESC node publishes by including the q-axis current data in the telemetry. This means that along with other motor parameters, the subscribers of the VESC state topic will now also get information about the torque-producing current. This data is crucial for applications that need to perform precise torque control or monitoring of the motor, enabling more sophisticated and responsive control algorithms.

8. **What is the purpose of modifying the arg name=”port” line in the vesc_driver_node.launch file?**

   The purpose of modifying the arg name="port" line in the vesc_driver_node.launch file is to configure the communication port that the VESC driver will use to connect to the VESC hardware. This modification is necessary when the hardware setup changes, such as when the VESC is connected to a different port on the Jetson Nano. By setting the correct port, you ensure that the driver can communicate with the VESC hardware, thereby enabling data exchange and control commands to be correctly sent and received.

9. **Explain the role of each component in the command $ rostopic pub -1/commands/motor/speed std_msgs/Float64 -- “7500.0”. What does each part mean?**

- ‘rostopic pub’: This command is used to publish data to a topic
- ‘-l’: message should only be published once
- ‘/commands/motor/speed’: name of the topic that is being published (the motor speed topic)
- ‘std_msgs/Float64’: specifies that the message being published is a float
- ‘—’: marks the end of the command and the beginning of the message
- ‘”7500.0”’: the message being published, this sets the motor speed to 7500 rpm

10. **Explain why is editing the CMakeLists.txt file and adding the catkin_install_python command an important step?**

    Editing the CMakeLists.txt file specifies what packages to install to ROS, adding catkin_install_python to the CMakeLists makes sure that the Python file executables are kept in the correct to be run by ROS.

11. **Explain the code in the controlled.launch file. In your explanation, include the following. Explain what the role of the Multiplier parameter is in JoyControl.py. What change occur when the multiplier is increased or decreased and how does it affect the duty cycle range? What is the control Method value and how does its value affect the AEV?**

    The controlled.launch file is written in the ROS XML format, meaning that there are tags with the corresponding information in between an opening an closing tag, or there are some parameters inside a tag.

    - <launch></launch>: this tag is used solely for wrapping other tags within it to create the launch file

- <include file=…/>: this is used to include other launch files within the current one. In this case, we are including the f710joy.launch file created last lab and the vesc_driver_node.launch file that sets up the vesc motor controller.

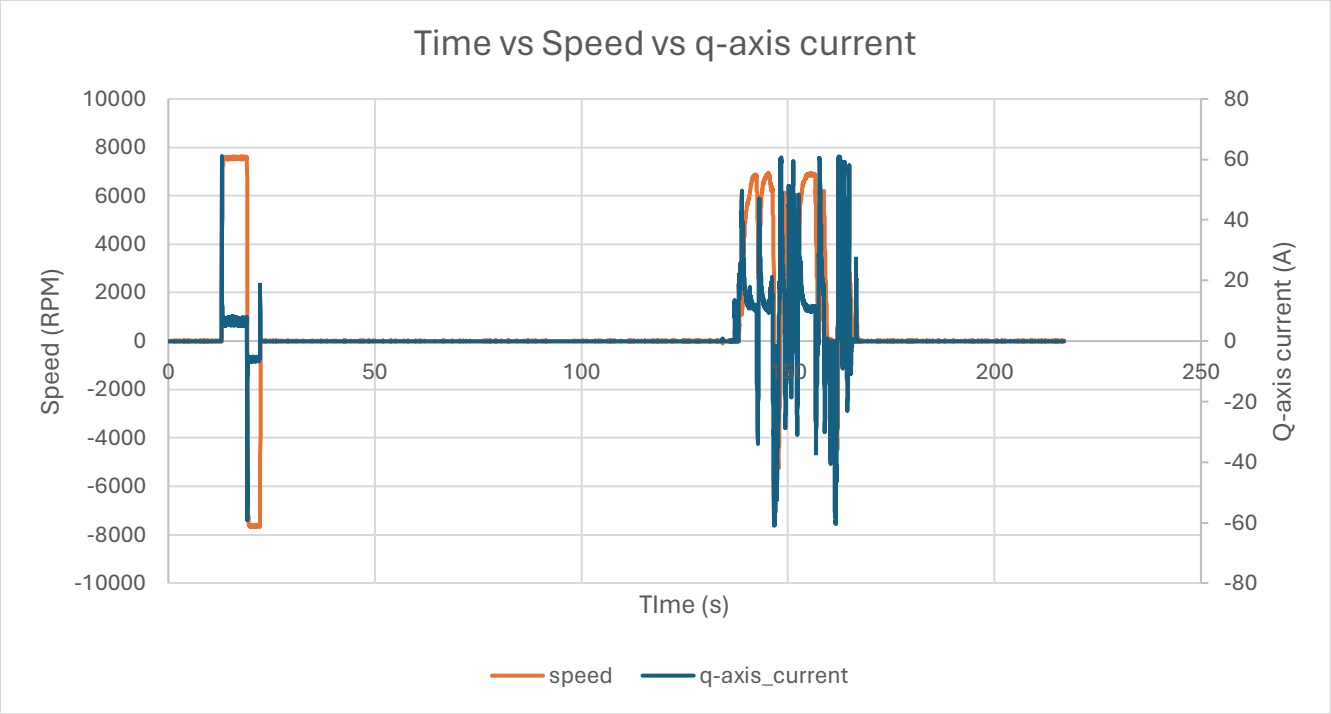
- <node name = "JoyControl" pkg = "test" type = "JoyControl.py" />: This line includes the node JoyControl from the package test, and attaches the executable to run which is the python file JoyControl.py when this node is used.

    - <param name = "Multiplier" value = "0.12" />: this line sends the Multiplier parameter to the JoyControl node, which then uses it in the JoyControl.py code. In the JoyControl.py code, this multiplier is multiplied to the right joystick value which ranges from -1 to 1, this multiplier changes this value to be from -0.12 to 0.12. This then goes directly into the duty cycle control command to then control the motor (0.12 duty cycle or -0.12 duty cycle – backwards). If the multiplier is increased there will be a higher duty cycle sent.
    - <param name = "ControlMethod" value = "0" />: this line sends the ControlMethod parameter to the JoyControl node, which in this case is 0. This selects the duty cycle control method in the JoyControl.py file, so that any value sent to the vesc is interpreted as a duty cycle for the motor speed. The values sent will be interpreted as a percent, so 0.5 is 50% speed.
- <node name = "servoControl" pkg = "test" type = "servoControl.py" />: This line includes the node servoControl with the executable servoControl.py. This file controls the turning of the AEV.

12. What value does the line a = -1*data.axes[0]*0.5+0.5 have within the callback function of servoControl.py? Consider the effect on the direction of the servo motor.

- This line calculates the value 'a' based on the input from the joystick.
- 'data.axes[0]' represents the position of the analog stick on the joystick. Multiplying by -1 reverses the direction of the control signal if needed.
- The term 0.5 is used to scale the input from the joystick to a suitable range for controlling the servo motor.
- Adding 0.5 shifts the range of values to ensure that the resulting value is within the appropriate range for controlling the servo motor.

13. Attach your graph plotted in Microsoft Excel in your final report and make sure you have your axis labeled for time, speed, and q-axis current. Explain your observations from the graph in detail. Comment on the relation between the speed and the q-axis current and explain your understanding thoroughly in at least 200 words. Provide examples from the lecture material on the operating principles of the motor to explain your answer.



Excel plot of Time vs Speed vs Q-axis Current

The torque produced by the motor is proportional to the product of the magnetic field strength, the current flowing through the windings, and the length of the conductor in the magnetic field. The motor's torque output can be controlled by varying the current in the stator windings. The plot above indicates a proportional relationship between the q-axis current and speed. This parameter directly controls the motor's torque output, thereby influencing its rotational speed. As the q-axis current rises, so does the torque generate by the motor. Through precise adjustments in the q-axis current, the motor controller can finely regulate the torque necessary to overcome resistive forces and enable smooth acceleration.

This control strategy, facilitated by Field-Oriented Control (FOC), allows for the independent regulation of torque and speed by manipulating the magnitude and phase angle of the currents in the stator windings. By aligning the stator current with the rotor magnetic field, FOC optimizes motor efficiency and performance, ensuring precise control across a wide range of speeds.

Elevated q-axis current may be essential at lower speeds to produce adequate torque for overcoming inertia and friction, thereby increasing rpm. The applied load on the motor can also influence the interplay between rpm and q-axis current. Under consistent load conditions, heightened q-axis current might translate to increased rpm as the motor compensates for the load torque. Nevertheless, this relationship may be more intricate in scenarios involving variable or dynamic loads, contingent upon the specific load profile.

Activity 5: What is a node in ROS?

A node really isn't much more than an executable file within a ROS package. ROS nodes use a ROS client library to communicate with other nodes. Nodes can publish or subscribe to a Topic. Nodes can also provide or use a Service.

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

Launch files are used to group multiple nodes with their respective parameters and run them together in one terminal. Instead of using rosrun where each node must be run in a different terminal with the given parameters manually inputted into the terminal.

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 attemted to run the rviz node, however since ROS was not running an error was given.

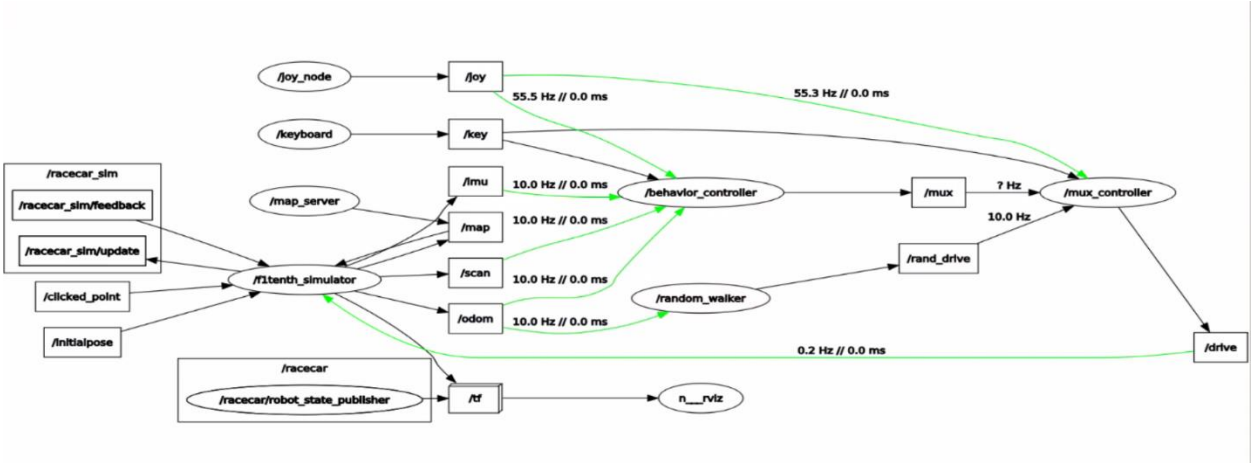
Activity 8: Briefly explain the role of each node launched by "simulator.launch" and relate them to their corresponding blocks in Fig. 1.

The f1tenth_simulator node and the rviz node is the Simulator block. The f1tenth_simulator is responsible for taking selecting the mode of control and and sends them to the rviz node which runs the simulation according to those parameters in the /drive topic. The behaviour controller block is the node with the same name. It selects what driver control mode to use to control the car with and sends the parameter data to the simulator block using the /drive topic. The MUX node/block takes input from the behaviour controller which contains the parameter data and passes on the appropriate input on the the /drive topic. The joy node (Joystick block) and the keyboard node (Keyboard block) take sensor data from the physical inputs and sends that data to the behaviour controller for selection and to the MUX for simulation. The random driver block (random_walker node) is a planning node which listens to sensor feedback from the simulation and sends back the same data after checking to see if the data breaks any constraints. If a constraint is hit, the type of constraint and that data is sent over to the simulation as well.

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

Topics are the information that nodes publish. Topics can be of different types, nodes will publish the actual topic and the type.

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.



The update rate of the /joy topic is determined by how fast information from the actual joystick can be received, as this might change rapidly, this rate is much higher to get the most updated position. The /imu, /map, /scan, /odom, and /rand_drive topics are published at an arbitrary rate as they are all internal. The /drive topic must be slower than the slowest topic published for no data to be missing, therefore it might be slower.

F1tenth_simulator nodes:
Publishes to: /joy, /key, /imu, /map, /scan, /odom, /tf, /racecar_sim/update
Subscribes: /racecar_sim/feedbcak, /clicked_point, /initialpose, /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?

The node that publishes to the **`/drive`** topic within the context of the F1TENTH simulator is used to send vehicle steering angle and linear velocity commands. The rate at which this node publishes, as well as the specific fields of **`/drive/drive`** that are being updated, are detailed in the simulator's documentation or configuration files. These fields typically represent the command messages for the vehicle's speed and direction, and may include timestamped information for synchronizing with other processes. To find the exact rate and the fields being updated, you would typically run a ROS (Robot Operating System) command to inspect the topic or look at the source code defining the publisher node's behavior.

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.

In electric vehicles, the modes of motor control such as speed and torque control play a crucial role in vehicle performance. Speed control mode adjusts the motor to maintain a specific speed, which is straightforward for drivers in manual mode but is also essential for maintaining consistent speed in self-driving mode.

Torque control mode, on the other hand, directly controls the torque output of the motor. This can be advantageous for electric vehicles as it provides precise control over acceleration, which is beneficial for both manual driving (especially in conditions requiring fine control such as slippery roads) and self-driving systems (where exact torque settings can be calculated and applied for smooth operation).

Each mode has its own benefits and can be used in different driving conditions and scenarios. Advanced control systems in electric vehicles can switch between these modes automatically to optimize performance, energy efficiency, and driver comfort, whether the vehicle is being manually driven or is in self-driving mode.

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

```
void publish_driver_command( const ros::TimerEvent&){

        double desired_delta = desired_steer_ang-last_servo;
double clipped_delta = std::max(std::min(desired_delta,
max_delta_servo), -max_delta_servo);
        double smoothed_servo = last_servo + clipped_delta;
last_servo = smoothed_servo;         servo_msg.data=smoothed_servo;

        double desired_rpm = desired_speed-last_rpm;
        double clipped_rpm = std::max(std::min(desired_rpm, max_delta_rpm),
max_delta_rpm);
        double smoothed_rpm = last_rpm + clipped_rpm;
last_rpm = smoothed_rpm;          erpm_msg.data=smoothed_rpm;

        servo_pub.publish(servo_msg);
        erpm_pub.publish(erpm_msg);
}
```

The "publish_driver_command" function is a part of the ROS program which is responsible for controlling the vehicle's steering and speed when driving the vehicle. It calculates the desired change in steering angle ("desired_delta") and speed ("desired_rpm") based on predefined values. These desired changes are then clipped to ensure they do not exceed maximum allowed rates of change ("max_delta_servo" for steering and "max_delta_rpm" for speed). The resulting values are smoothed by adding them to the last known values, and then these smoothed values are published to respective ROS topics using ROS publishers ("servo_pub" and "erpm_pub").

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

```
  void laser_callback(const sensor_msgs::LaserScan & msg) {

        int n1=msg.ranges.size()/2;
        std::vector<float> scan_ranges=msg.ranges;
std::vector<float> scan_intensities=msg.intensities;         for
(int i=0;i<n1;i++){             scan_ranges[i]=msg.ranges[n1+i];
scan_ranges[n1+i]=msg.ranges[i];
scan_intensities[i]=msg.intensities[n1+i];
scan_intensities[n1+i]=msg.intensities[i];
        }

       // Publish lidar information
sensor_msgs::LaserScan scan_msg;
scan_msg.header.stamp = msg.header.stamp;
scan_msg.header.frame_id = msg.header.frame_id;
scan_msg.angle_min = msg.angle_min+3.141592;
scan_msg.angle_max = msg.angle_max+3.141592;
scan_msg.angle_increment = msg.angle_increment;
scan_msg.range_max = msg.range_max;
scan_msg.ranges = scan_ranges;
scan_msg.intensities = scan_intensities;
lidar_pub.publish(scan_msg);
    }
```

The “laser_callback” function is designed to handle incoming laser scan data in the ROS environment. Once a laser scan message is received (“msg”), the function first calculates the midpoint index “n1” of the scan data array. Then, it swaps the first half of the ranges and intensities data with the second half, which reverses the order of the values. After swapping, a new “sensor_msgs::LaserScan” message (“scan_msg”) is created, which becomes the header from the received message. The angle parameters of the scan message (“angle_min”, “angle_max”, and “angle_increment”) are adjusted by adding π to their original values, effectively shifting the scan angles by 180 degrees. The maximum range value remains unchanged. Finally, the modified scan ranges and intensities are assigned to “scan_msg” and published to its respective ROS topic using “lidar_pub”. This process is the same as the received laser scan data and publishes it.

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

```
zoomzoom@zoomzoom-desktop:~/catkin_ws$ rosnode list
/behavior_controller
/joy_node
/keyboard
/mux_controller
/racecar_experiment
/rosout
/rplidarNode
/vesc_driver_node
zoomzoom@zoomzoom-desktop:~/catkin_ws$ rostopic list
/brake
/brake_bool
/commands/motor/brake
/commands/motor/current
/commands/motor/duty_cycle
/commands/motor/position
/commands/motor/speed
/commands/servo/position
/diagnostics
/drive
/imu
/joy
/joy/set_feedback
/key
/mux
/nav
/odom
/rand_drive
/rosout
/rosout_agg
/scan
/scan2
/sensors/core
/sensors/servo_position_command
```

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?

```
/sensors/servo_position_command
zoomzoom@zoomzoom-desktop:~/catkin_ws$ rostopic hz /commands/servo/position
subscribed to [/commands/servo/position]
average rate: 75.335
        min: 0.008s max: 0.019s std dev: 0.00142s window: 73
average rate: 75.201
```

Some factors that could affect it are hardware capabilities.

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

Kw = K * w(t) w(t) = we
We = wm*pp
Wm = RPM*(pi/30)

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 line of code from experiment.cpp sets up a timer in a ROS (Robot Operating System) node. It uses the createTimer function to regularly call the publish_driver_command method of the RacecarExperiment class. The timer duration is set to the inverse of driver_smoother_rate, which means the publish_driver_command function is called at a frequency equal to driver_smoother_rate. The this pointer is used to pass the current instance of the RacecarExperiment class to the callback function, allowing it to access the member variables and functions of the class. This pattern is typically used to periodically execute a task in ROS, such as publishing a command to control a racecar in a simulation or real-world experiment.

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

```
$ rosrun
$ roslaunch
$ rosnode
$ rostopic
```

rosrun: This command runs a specific node within a ROS package. Executing a single node directly without launching an entire ROS system is handy.
roslaunch: This command is used to launch one or more ROS nodes along with their required parameters and configuration files. It helps start complex systems consisting of multiple nodes.
rosnode: This command-line tool provides information about ROS nodes, such as listing active nodes or getting information about a specific node. A use case could be to check which nodes are running or troubleshoot issues related to node communication
rostopic: This command-line tool is used to interact with ROS topics. It provides functionalities like listing active topics, displaying published messages, or manually publishing messages to a topic for testing purposes.

Activity 1: Derive the homogenous transformations that relate the frame "base_link" to the frames "laser", "imu", and "camera", i.e., $Tlaser\ base\_link$, $Timu\ base\_link$, $Tcamera\ base\_link$.

A = 90 deg
Timu base_link = [12.86 0 55]
Translated$_z$ = 55 + 6.19 = 61.19mm
X = 12.86mm

$R_z(a) = \begin{bmatrix} cosx & -sinx & 0 \\ sinx & cosx & 0 \\ 0 & 0 & 1 \end{bmatrix}$

$T = A_{Timu\ base\_link} = \begin{bmatrix} 0 & -1 & 0 & 12.86 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 55 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

qw = 1/2*sqrt(1+0+0+1) = sqrt(2)/2

$qx = \frac{0-0}{4\frac{sqrt(2)}{3}} = 0$ $qy = qz = \frac{0-0}{4w} = 0$

0.707, 0, 0, 0.707

therefore, $[\frac{sqrt(2)}{2}, 0,0, \frac{sqrt(2)}{2}]$

A = 180 deg
Timu base_laser = [12.86 0 76]
Translated$_z$ = 55 + 21 = 76mm
X = 12.86mm

$R_z(a) = \begin{bmatrix} cosx & -sinx & 0 \\ sinx & cosx & 0 \\ 0 & 0 & 1 \end{bmatrix}$

$T = A_{Timu\ base\_link} = \begin{bmatrix} -1 & 0 & 0 & 12.86 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 76 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

qw = 1/2*sqrt(1-1-1+1) = 0

qz = 1 $qy = qx = \frac{0-0}{4w} = 0$

therefore, $[0,0,0,1]$

Activity 2: Complete the following ROS static transformation in the "experiment.launch" file that is provided to you on Avenue to Learn. The information given in Figs. 1&2 is needed for performing this task. Explain the purpose of these transformations.

The first transformation, "base_link_to_imu", defines the positioning and orientation of an IMU sensor ("imu") with respect to the robot's base frame ("base_link"). The second transformation, "base_link_to_laser", defines the relationship between the robot's base frame and a laser sensor ("laser"). These transformations, published by the "static_transform_publisher" node, provide information about position for tasks like sensor fusion, localization, and navigation, which is done later in the lab, within the ROS ecosystem. The values in the "args" variable need to be filled with specific values based on the figures provided in the lab manual.

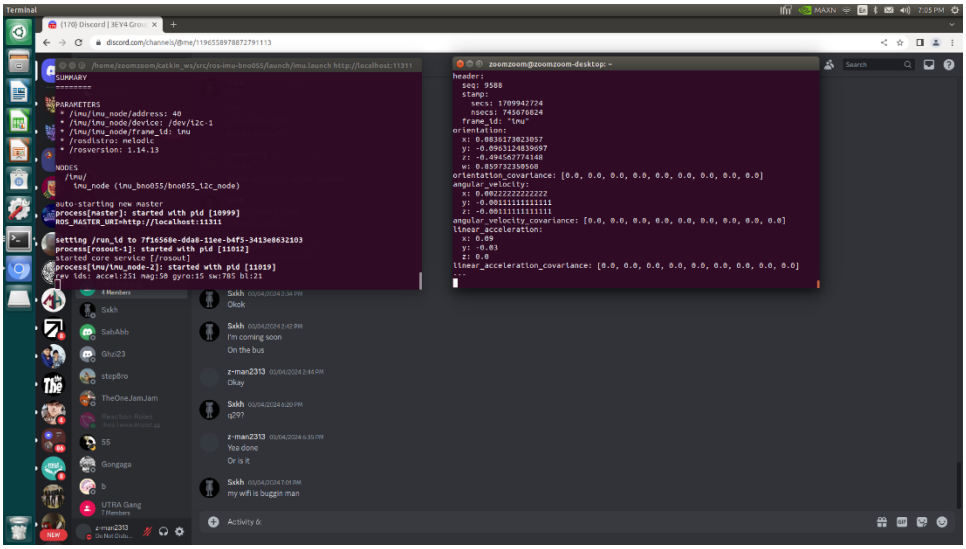
Activity 3: Refer to the course lecture notes to briefly explain how the orientation of a frame is expressed in Quaternion vector.

The Quaternion vector represents a rotation in space. The vector contains four components, 1 scalar (real) and 3 imaginaries (complex).

The Scalar represents the amount of rotation around the axis (magnitude of rotation). The imaginary specifies the axis of rotation and the direction of rotation in 3D space.

Activity 6: It is usually a good idea to calibrate the IMU before utilizing its data. Follow the procedure outlined on Page 51 of the sensor data sheet in order to calibrate it. Echo the value of "Calibration status" on the terminal screen as you move the vehicle manually. What status value would indicate a fully calibrated IMU?

A fully calibrated imu would have a calibration status of 255.



Activity 8: The ROS node imu_tf_pub.cpp written in C++, the corresponding launch file imu_tf_pub.launch, and the file CMakeLists.txt needed to build the package are provided to you on Avenue to Learn. Create a ROS package named imu_tf_pkg within your catkin workspace (see here for instructions) with the following dependencies: roscpp std_msgs sensor_msgs tf. Next, add the file imu_tf_pub.cpp as a node to your ROS package. Examine this file's content and briefly explain in your report what the new node does. Add the imu_tf_pub.launch file to the launch folder of the package, replace the CMakeLists.txt file in the package with the one you downloaded from avenue, and build the package.

This node listens to IMU data published on a specified topic, converts the orientation data into a transform, and broadcasts this transform, allowing other nodes in the ROS system to use the IMU orientation data for various purposes such as robot navigation and localization.

Activity 13: Do you observe any errors in the wheel odometry-based localization in your experiment? Explain. Discuss potential sources of errors in wheel-based odometry computations that may cause a drift in the computed position and orientation of the vehicle.

Wheel slippage is a reason behind error in the calculated position. When the wheel slips or loses traction, it results in inaccurate measurements of distance traveled, leading to discrepancies in the calculated position.

In situations where the wheels skid or slide, such as during sudden braking or sharp turns, the actual distance traveled differs from what is calculated based on wheel rotations.

Noise in sensor measurements, such as encoder readings or gyroscopic sensors used for orientation estimation, introduces errors in odometry calculations.

Errors in the initial localization or mapping of the environment propagate throughout the localization process.

Activity 16: Compare the results of vehicle localization with and without using the IMU yaw angle measurement. This can be observed by comparing the frames "base_link" and "base_link_imu" in rviz visualization tool, while you are driving the vehicle around. You should set the rviz fixed frame to "odom". As in the previous case, you should drive back the vehicle to its position/orientation and observe any accumulated error in localization. Explain how the code in "experiment.cpp" handles the difference in the frame of reference for measuring the yaw angle between IMU (E-N-U frame) and the wheel odometry "odom". Do you still expect to see localization error accumulation even after using the IMU data? Why?

Even after incorporating IMU data, some localization error accumulation may still occur. Wheel odometry is prone to cumulative errors due to wheel slippage and calibration inaccuracies. While the IMU provides orientation data, it may not completely mitigate the drift caused by wheel odometry errors. Factors such as sensor noise, calibration errors, and integration algorithms can also contribute to residual errors in localization, although potentially reduced compared to relying solely on wheel odometry.

Activity 18: Explain the roles of the parameters "use_odom", "use_imu", "kf_dist_linear" and "kf_dist_angular" in the Scan Matcher algorithm. You can consult the course lecture materials and the documentation and resources for the algorithm.

- use_odom:
    - This parameter shows whether to use odometry data in the Scan Matching algorithm. If this parameter is set to true or used, the algorithm incorporates odometry information to improve the accuracy. If set to false or not used, odometry data is not used, and the algorithm relies solely on the sensor readings.

- use_imu:
    - Like "use_odom," this parameter determines whether IMU data should be used in the Scan Matching algorithm. IMU sensors provide information about the robot's orientation and acceleration. This parameter allows the algorithm to use IMU data for better accuracy, mainly in scenarios where odometry alone may not be as efficient.
- kf_dist_linear (Keyframe Distance Linear):
    - This parameter allows you to set the distance the fixed frame needs to move before updating the keyframe scan. The change in pose is calculated using the distance between the current laser scan and a "keyframe" scan. The keyframe scan is updated after the robot moves a certain distance. So, if the robot stands still, the keyframe scan will not change, and the pose will remain more drift-free.
- kf_dist_angular (Keyframe Distance Angular):
    - Almost like "kf_dist_linear," this parameter allows you to set the angle at which the fixed frame needs to move before updating the keyframe scan.

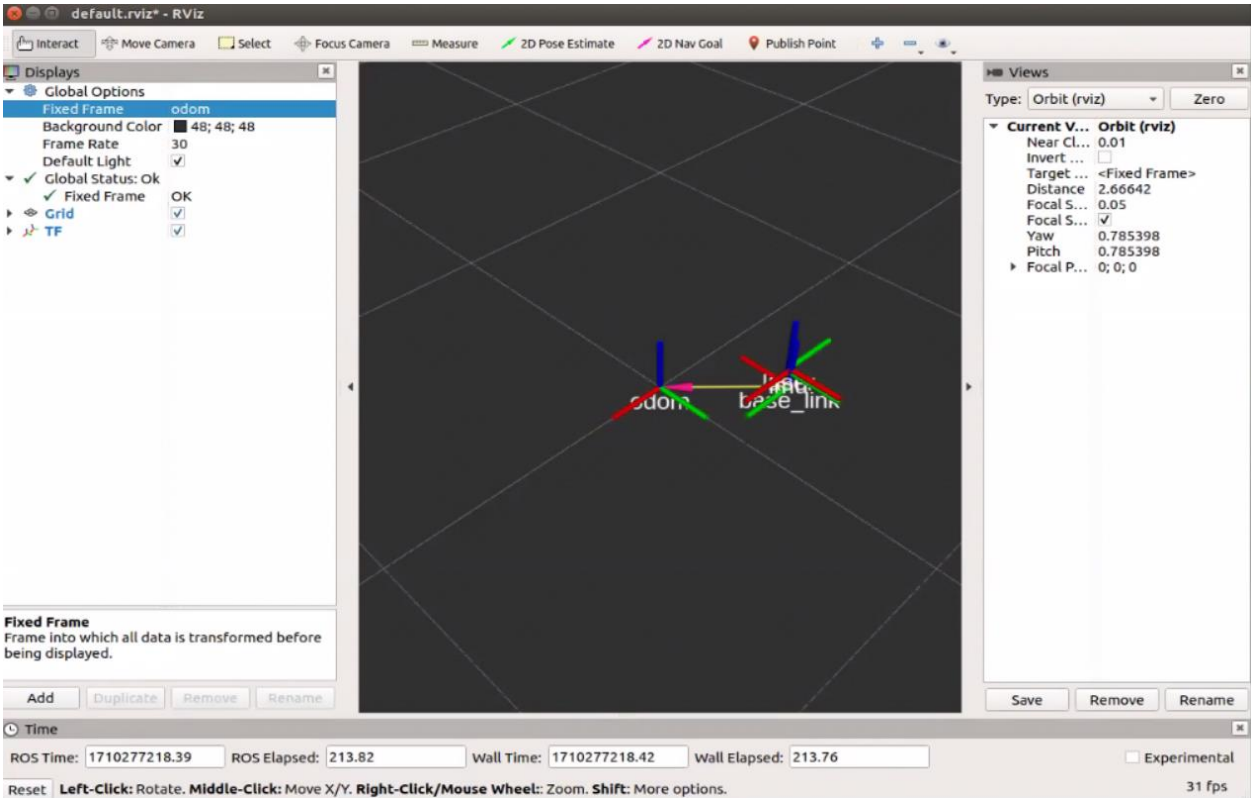
Activity 19: Locate and open the file "laster_scan_matcher.cpp" in the scan tools package. Referring to the lecture notes on Localization and Mapping, the homogenous transformations $T_{b_k}^{\circ}$, T_{key}° and T_{l}° are represented by the variables "f2b_", "f2b_kf_", and "laser_to_base_" in this file. Examine the following the lines of code and explain what they do mathematically.

These lines of code are part of a process that involves updating the position and orientation of a base frame relative to the world frame, incorporating corrections based on sensor measurements or calibration data.

The first line calculates the correction of the base's position within its own frame. It combines transformations to convert the correction from the base frame to the laser frame (where some measurements or corrections may have been made), and then back to the base frame. This step ensures that the correction is applied correctly within the base frame's reference frame.

The second line updates the pose of the base frame in the world frame by applying the calculated correction. It combines the current pose of the base frame relative to the world frame with the previously computed correction in the base frame. This update ensures that any adjustments made to the base frame's position within its own frame are reflected accurately in its position relative to the world frame.

Activity 21: Launch the file "experiment.launch". Use ROS visualization tool, rviz, to explore the performance of the localization algorithm. Set the fixed frame to "odom" and add a TF type display from the left panel. Also add a LaserScan type display with the topic "scan" to show the laser scan data. Observe how the frame "base_link" moves with the respect to the frame "odom" as you drive the vehicle around. Pay attention to the segments of the drive where the vehicle surrounding environment may have uniform features, e.g., when driving along two parallel walls. Would Scan Matcher by itself without wheel odometry and IMU would be effective in such case? Compare the performance of the Scan Matcher algorithm in localization with and without this data. Record the result for use in the report.



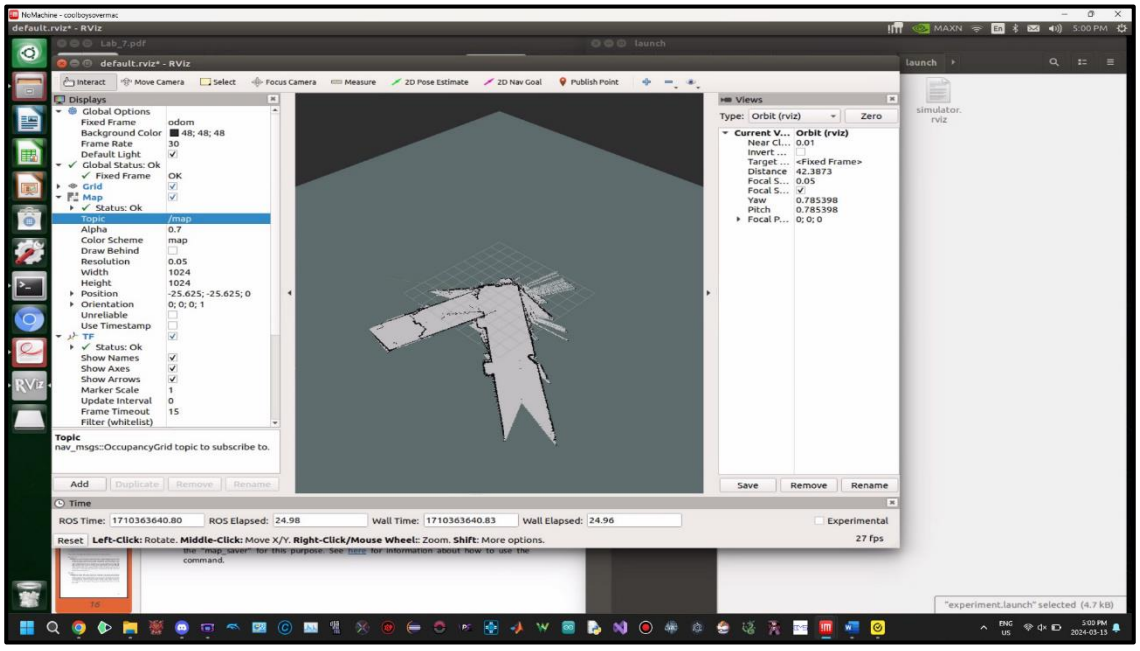
Activity 22: Add a "LaserScan" display type and set its topic to "scan". This should allow you to observe the laser scan data while driving the vehicle. Observe and compare the laser scan data in the two cases: (i) the rviz display (fixed) frame is set to "base_link" (ii) the rviz display (fixed) frame is set to "odom". Explain what is happening in case (ii). How would you transform the LiDAR scan data to the fixed frame?

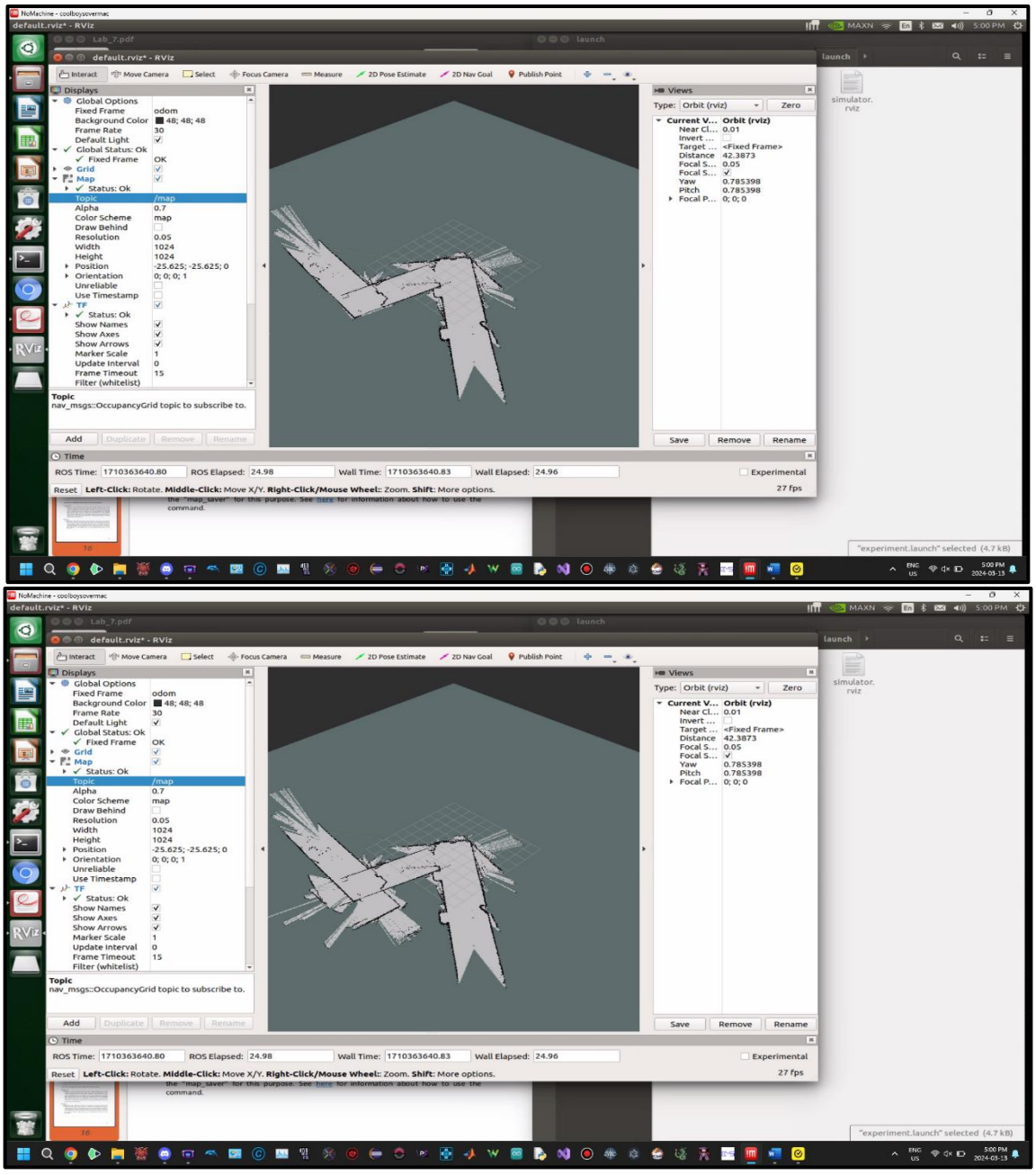
When the fixed frame is set to "odom," RViz visualizes the laser scan data in the context of the odometry frame, which represents the world or global frame of reference. This means that the laser scan data is visualized as if it were observed from the perspective of an external observer looking at the robot's motion within the environment.

the laser scan data appears stationary in RViz relative to the world frame ("odom"). As the AEV moves, the laser scan data appears to remain fixed in the environment, while the AEV's frame ("base_link") moves relative to it.

To transform the LiDAR scan data to the fixed frame ("odom"), a coordinate transformation must be applied. This transformation involves using the AEV's pose (position and orientation) relative to the fixed frame and applying it to the LiDAR scan data. Specifically, the laser scan data, initially expressed in the LiDAR's frame, is transformed into the fixed frame ("odom") using a combination of the AEV's pose (obtained from wheel odometry, IMU, or other localization methods) and the transformation between the LiDAR's frame and the robot's frame.

Activity 24: Ensure that the vehicle surrounding is clear of objects and people as you will be driving the vehicle using the Joystick. Launch the modified "experiment.launch" file to start the SLAM algorithm. Also launch the ROS visualization tool rviz to display the results of the SLAM algorithm. Set the rviz fixed frame to "map" and add displays of TF and Map types from the left panel. Deselect all frames except for map (fixed frame) and base_link (body-attached frame) under TF. To display the map data, add "map" under the topic option of Map display. Drive the vehicle around slowly and observe how the map of environment is constructed while the vehicle is simultaneously localized in this map. Comment on what you observe.





From these 3 figures above we can see that the robot was able to map out the hallway as seen by the long rectangular shapes and whenever the angle was changed by a certain degree a new scan was completed allowing us to see the hallway again but this time when it makes a turn to one of its sides.

Activity 1: Explain why this might be a reasonable assumption based on the configuration of the sensors on MacAEV. With this assumption, show the angle α_i can be computed from the angle information in LiDAR scan data.

Neglecting the displacement between the "laser" frame (the LiDAR sensor's position) and the "base_link" frame is a justifiable approach owing to their close alignment, rendering the impact of displacement on obstacle detection practically insignificant. The angle \(\alpha_i \), delineating the orientation between an obstacle's closest point and the x-axis of the "base_link" frame, can be readily derived from LiDAR's angular data. This streamlines the incorporation of LiDAR data into the collision avoidance algorithm, utilizing the reported angle as \(\alpha_i \) to denote the obstacle's orientation relative to the vehicle's forward direction.

Activity 2: Derive the above expression for δ_o^{k+1} in (17). Also, explain the role of the design parameter $0 \leq \eta_\delta \leq 1$.

The value of Ndelta is affected by the force applied to the joystick. When Ndelta equals 0, the vehicle is manually driven, with commands originating from the joystick. Conversely, when Ndelta equals 1, the algorithm assumes control, facilitating autonomous driving. If the AEV. If at the minimum distance, the transition of the Ndelta from 0 to 1 may be delayed.

Delta O comprises the angles between the base link and detected obstacles. These angles serve as the basis for steering angle adjustment. Specifically, the steering angle is determined to oppose the direction of the detected obstacle.

Activity 3: Copy the content of this file into a new file named “collision_assistance.py” in the directory “node” of f1tenth_simulator software stack. Make this file executable by running the following command at the terminal:

Executing the command chmod +x collision_assistance.py renders the file executable. Please note that this command is limited in its execution, requiring the addition of the keyword sudo beforehand.

Activity 4: The following function finds potentially dangerous obstacles and their distance to the vehicle. Complete the missing code in "…" to achieve this functionality:

```
# Find unsafe obstacles and return number of unsafe obstacles
#  and the closest point of each unsafe obstacle to the vehicle
def obst_idnt(self,ls_ranges):

    indx=np.zeros((2),dtype=int)
    j=0
    nm_obs=0

    for i in range(self.ls_len_mod2):
        if ls_ranges[i,0]<=self.d_obs and i<self.ls_len_mod2-1:
            if j==0:
                indx[0]=i
                j=1
            indx[1]=i+1 #
        else:
            j=0
            if indx[1]-indx[0]>0:
                self.obs_indx[nm_obs,:]=indx
                nm_obs=nm_obs+1
                indx[1]=0;indx[0]=0

    arg_ls_obs=np.zeros(nm_obs,dtype=int)

    for i in range(nm_obs):
        arg_ls_obs[i]=np.argmin(ls_ranges[self.obs_indx[i,0]: self.obs_indx[i,1],0]) #
        arg_ls_obs[i]=self.obs_indx[i,0]+arg_ls_obs[i] #

    return nm_obs, arg_ls_obs
```

Activity 5: The function "ptn_fld" computes the total obstacles-related corrective forces acting the vehicle. Complete the missing code in "…" to achieve this functionality:

```
# compute corrective force acting on the vehicle
def ptn_fld(self, nm_obs,arg_ls_obs,ls_ranges):

    f_rep_x=0;f_rep_y=0

    for i in range(nm_obs):
        d=ls_ranges[arg_ls_obs[i],0]
        theta=ls_ranges[arg_ls_obs[i],1]

        f_rep_x=f_rep_x+self.f_gain*(1-self.d_obs/d)*math.cos(theta) #
        f_rep_y=f_rep_y+self.f_gain*(1-self.d_obs/d)*math.sin(theta) #

    return f_rep_x, f_rep_y
```

Activity 6: Find and complete the following lines of code in the function "lidar_callback". Explain the purpose of this part of the code:

```
def lidar_callback(self, data):
    ranges = data.ranges
    proc_ranges = self.preprocess_lidar(ranges)
    nm_obs,arg_ls_obs=self.obst_idnt(proc_ranges)

    # Calculate the potential field forces
    f_tot_x, f_tot_y=self.ptn_fld(nm_obs,arg_ls_obs,proc_ranges)

    # Calculate desired velocity and steering angle
    self.vel_x=self.vel+f_tot_x

    vel_d= self.etha_vel * self.vel_joy + (1-self.etha_vel) * self.vel #

    if self.vel_joy >=0 and vel_d < 0:
        vel_d = 0

    u0=self.etha_delta*math.tan(self.steer_joy)+ (1-self.etha_vel) * f_tot_y * self.wheelbase/max(abs(self.vel),0.001) ** 2 #
    delta_d=math.atan((u0-math.tan(self.steer_joy))/(1+u0*math.tan(self.steer_joy)))+self.steer_joy

    if delta_d >=self.max_steering_angle:
        delta_d=self.max_steering_angle
    elif delta_d<=-self.max_steering_angle:
        delta_d=-self.max_steering_angle
    if vel_d>=self.max_speed:
        vel_d=self.max_speed
    elif vel_d<=-self.max_speed:
        vel_d=-self.max_speed
```

This code snippet is part of a ROS (Robot Operating System) program designed for an autonomous electric vehicle. In the `lidar_callback` function, it processes data received from the LiDAR sensor mounted on the vehicle. Initially, it extracts the range data from the sensor readings and preprocesses it using a method called `preprocess_lidar`. Then, it identifies obstacles and their corresponding angles using the `obst_idnt` function. Following this, the code calculates the potential field forces based on the identified obstacles and range data through the `ptn_fld` function. These forces are used to determine the desired velocity and steering angle for the vehicle's autonomous navigation. Notably, it blends the desired velocity obtained from both a joystick input (`vel_joy`) and the vehicle's current velocity (`vel`). Additionally, it computes the desired steering angle (`delta_d`) considering the vehicle's current steering angle from the joystick (`steer_joy`) and potential field forces. Finally, it applies constraints on the desired velocity and steering angle to ensure they do not exceed predefined maximum values for safe operation.

Activity 9: Examine the code in "collision_assistance.py" and relate the remaining parameters to those used in the collision avoidance assistance algorithm described above. You can tune these parameters to achieve a desired response from the vehicle in the simulation environment. Drive the vehicle in the virtual environment with and without collision avoidance assistance and notice any difference in its behavior. Use the simulation to explore the impact of the above parameters on the vehicle response. Include your observations in your report.

The parameters `distance_to_obstacle_th`, `force_gain`, `velocity_correction_gain`, and `steering_correction_gain` directly influence the collision avoidance assistance algorithm in "collision_assistance.py". `distance_to_obstacle_th` sets the threshold distance to activate the algorithm, while `force_gain` determines the strength of corrective maneuvers to avoid collisions.

Additionally, `velocity_correction_gain` and `steering_correction_gain` regulate adjustments to the vehicle's speed and steering angle when navigating obstacles. Meanwhile, the `collision_assistance_button_idx` parameter designates the "X" button on the joystick to toggle the algorithm, enabling or disabling collision avoidance assistance. During simulation, activating the algorithm leads to cautious navigation, avoiding obstacles, whereas deactivating it relies on manual control. By fine-tuning these parameters and observing their effects, users can optimize collision avoidance performance for safe and effective autonomous navigation in the simulation environment.

Activity 11: Examine the vehicle behavior in experiment. Open "experiment.launch" file and make sure that the collision assistance avoidance node is launched. Drive the vehicle around with collision avoidance assistance activated and evaluate its response. Fine tune the parameters of the algorithm as needed and observe the impact of these parameters on the vehicle response. Include your observations in your written report.

After executing the experiment.launch file, we activated the collision avoidance feature. We noted that the car autonomously veered away from walls when nearing them. Nevertheless, we observed that if the car moved too swiftly or navigated through cramped spaces, the collision avoidance mechanism failed to function adequately due to insufficient time or space for maneuvering.

Activity 12: Some of the mathematical derivations in the rest of this document are intentionally left incomplete. You must complete and include these derivations in your report.

dlr = -(Vs^2/l) *(cos(ar) – cos(al))*tan(delta)

Deltalr = atan(-l/(Vs^2*(cos(ar) – cos(al)) * (-kp*der-kd*de))

Hc1(s) = D(s)/Ddes(s) = kp/s^2+kd*s+kp

Alphal = 3pi/2 – betal – thetal - <al

dr= br * cos(betar)

dl = bl * cos(betal)

Activity 13: Determine the steady-state tracking error in response to a constant distance command in the closed-loop control.

Steady state error is given by,

$$\lim_{s\to 0} sH_{cl}(s) = \lim_{s\to 0} s\frac{k_p}{s^2+k_d s+k_p} = 0$$

Since the constant input is like a unit step input when it is inputted, since the transfer function is a 2nd order, the ess is expected to be 0.

Activity 16: Explain what the following section of the code accomplishes.

```
min_distance = min(data.ranges[-sec_len+int(self.scan_beams/2):sec_len+int(self.scan_beams/2)])
velocity_scale = 1-math.exp(-max(min_distance-self.stop_distance,0)/self.stop_distance_decay)


velocity=velocity_scale*self.vehicle_velocity
```

In general, this code calculates a scaling factor for the velocity of the car based on the proximity of obstacles detected by the sensors, the velocity therefore decreases as obstacles come closer.

Looking at the first line, it calculates the minimum distance from a set of sensor readings that are provided by the variable “data.ranges”. This calculation is used to find the closest obstacle within a certain range. The second line is a little more complex, as this line calculates a scaling factor for the velocity based on the minimum distance to any obstacle.

- “min_distance-self.stop_distance”: This calculates the difference between the minimum distance to obstacles and a predefined stop distance
- “max(min_distance-self.stop_distance,0)”: If “min_distance” is greater than or equal to “self.stop_distance”, this difference will be positive; otherwise, it will be defaulted to zero
- “max(min_distance-self.stop_distance,0)/self.stop_distance_decay”: Now we divide by this factor to scale the positive difference in distance by a decay factor
- “1 - math.exp(...)”: The we subtract the exponential of the scaled difference from 1. From this we can understand that the scaling factor will be close to 1 when the robot is far from obstacles and closer to 0 when it's close to obstacles

Now finally looking at the last line, we can see that it multiplies the scaling factor of the velocity that was just calculated in the line above with the base velocity of the car. This line is used to adjust the velocity of the vehicle based on the proximity of any obstacles, reducing speed as obstacles come closer for more precise control.

Activity 19: Do you observe any difference in the vehicle behavior between the simulations and the experiment? Comment on your observations in this regard.

During the simulation, the performance appeared significantly smoother compared to real-world conditions. When operating the AEV in real-life scenarios, numerous machine errors were encountered. Additionally, the LiDAR system utilized only two points in the scanning algorithm, indicating an enhanced algorithm's need to optimize autonomous driving capabilities.

Activity 2: Study the files “navigati on_lab9_inc.py” and “params.yaml” and identify parts of the code that accomplish this step.

```
168    safe_distance: 2.0
169    right_beam_angle: rad(2*pi*4.0/16)
170    left_beam_angle: rad(2*pi*12.0/16)
```

Above parameters used to define FOV in params.yaml.

```
# Define field of view (FOV) to search for obstacles

self.ls_str=int(round(self.scan_beams*self.right_beam_angle/(2*math.pi)))
self.ls_end=int(round(self.scan_beams*self.left_beam_angle/(2*math.pi)))
self.ls_len_mod=self.ls_end-self.ls_str+1
self.ls_fov=self.ls_len_mod*self.ls_ang_inc
self.angle_cen=self.ls_fov/2
self.ls_len_mod2=0
self.ls_data=[]
```

The code above is defining variables to restrict the angle of interest when creating the FOV to 180 degrees in front of the car.

```
    # Pre-process LiDAR data
    def preprocess_lidar(self, ranges):

        data=[]
        data2=[]

        for i in range(self.ls_len_mod):
            if ranges[self.ls_str+i]<=self.safe_distance:
               data.append ([0,i*self.ls_ang_inc-self.angle_cen])
            elif ranges[self.ls_str+i]<=self.max_lidar_range:
                data.append ([ranges[self.ls_str+i],i*self.ls_ang_inc-self.angle_cen])
            else:
                data.append ([self.max_lidar_range,i*self.ls_ang_inc-self.angle_cen])

        k1 = 100
        k2 = 40

        for i in range(k1):

            s_range = 0

            for j in range(k2):
                index1 = int(i*len(ranges)/k1+j)
                if index1 >= len(ranges):
                    index1 = index1-len(ranges)

                index2 = int(i*len(ranges)/k1-j)
                if index2 < 0:
                    index2= index2 + len(ranges)

                s_range = s_range + min(ranges[index1],self.max_lidar_range)+ min(ranges[index2],self.max_lidar_range)

            data2.append(s_range)
```

The preprocess_lidar function takes these variables and uses them to select only data that is within the FOV range and ignore everything else.

The navigation.py and params.yaml files in Lab 9 establish the field of view (FOV) in front of the vehicle. Parameters such as right_beam_angle, left_beam_angle, and scan_beams are defined to facilitate the determination of the FOV. As depicted in the preceding figure, the effective range of the scan beams is calculated by multiplying the number of beams with the angular ranges for both left and right angles. Subsequently, to obtain the number of beams (ls_len_mod), the difference between ls_end and ls_str is computed and incremented by one to account for both endpoints. Finally, the FOV (ls_fov) is calculated by multiplying the number of scan beams by the angular increment.

Activity 3: Explain the role of function “preprocess_lidar” and how it helps accomplish the objective of Step 2.

```
98      # Pre-process LiDAR data
99      def preprocess_lidar(self, ranges):
100
101         data=[]
102         data2=[]
103
104         for i in range(self.ls_len_mod):
105             if ranges[self.ls_str+i]<=self.safe_distance:
106                data.append ([0,i*self.ls_ang_inc-self.angle_cen])
107             elif ranges[self.ls_str+i]<=self.max_lidar_range:
108                 data.append ([ranges[self.ls_str+i],i*self.ls_ang_inc-self.angle_cen])
109             else:
110                 data.append ([self.max_lidar_range,i*self.ls_ang_inc-self.angle_cen])
111
112         k1 = 100
113         k2 = 40
114
115         for i in range(k1):
116
117             s_range = 0
118
119             for j in range(k2):
120                 index1 = int(i*len(ranges)/k1+j)
121                 if index1 >= len(ranges):
122                     index1 = index1-len(ranges)
123
124                 index2 = int(i*len(ranges)/k1-j)
125                 if index2 < 0:
126                     index2= index2 + len(ranges)
127
128                 s_range = s_range + min(ranges[index1],self.max_lidar_range)+ min(ranges[index2],self.max_lidar_range)
129
130             data2.append(s_range)
```

The preprocess_lidar function receives input data from the LiDAR and transforms it into an array format. This array consists of two rows: the first row contains the distance from the LiDAR to the object, while the second row represents the angle of that measurement relative to the LiDAR's starting point. It also ignores any data that is not in the vehicles defined FOV that the LiDar receives.

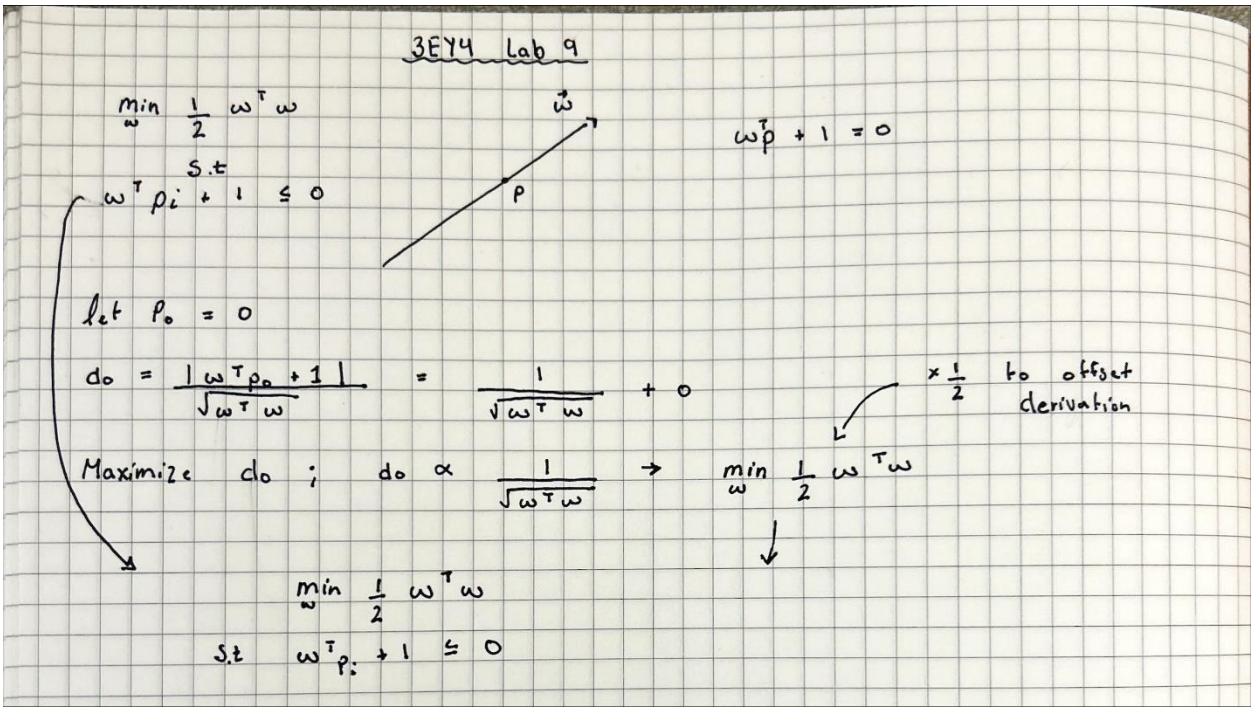
Activity 5: Complete the code for the function "find_best_point" to compute the desired direction of movement according to the formulation in (2). Explain why this might be a better choice than the furthest point in the largest gap in (1).

We begin by initializing the variables "best_heading" and "range_sum" to 0. Then, we employ a For loop to iterate over each index within the specified range from start_i to end_i. This loop enables us to traverse through the gap effectively. At each index in this gap, the product of distance and angle is added to the "best_heading" sum, while the distance itself is accumulated in "range_sum". Subsequently, after the loop, we divide "best_heading" by "range_sum" to determine the desired angle for the vehicle's direction.

```
169     # start_i & end_i are the start and end indices of max-gap range, respectively
170     # Returns index of best (furthest point) in ranges
171     def find_best_point(self, start_i, end_i, proc_ranges):
172
173         range_sum = 0
174         best_heading = 0
175
176         for i in range (start_i,end_i + 1):
177         range_sum += proc_ranges[i,0]
178         best_heading += proc_ranges[i,0]*proc_ranges[i,1]
179
180     if range_sum != 0:
181         best_heading = best_heading/range_sum
182
183
184         return best_heading
```

An if-else statement is employed to handle a scenario where "range_sum" equals 0. This function facilitates the selection of a direction for the vehicle within a gap based on a weighted average. This code has been implemented and is visible in the preceding figure.

Activity 6: Derive the optimization formulation in (7).



Activity 7: Explain how the value of $0 \leq \alpha \leq 1$ affects the solution to the optimization problem in (8). Consider what happens as α changes between the two extreme values of $\alpha = 0$ and $\alpha = 1$.

The parameter α, where $0 \leq \alpha \leq 1$, serves as a design parameter. Using the formula, we aim to balance maximizing the distance of the separating barrier from the origin and minimizing the change in the virtual barrier between time steps k – 1 and k.

When $\alpha = 0$, The second term in the minimization expression becomes zero, as α is multiplied by 0.

When $\alpha = 1$, The first term in the minimization expression becomes zero, as $(1 - \alpha)$ becomes 0.

Overall, as α varies between 0 and 1, the weight assigned to the two terms in the minimization expression changes. When $\alpha = 0$, the focus lies solely on wk, while when $\alpha = 1$, the emphasis shifts entirely to the term involving ans - ans. For values between 0 and 1, the trade-off between the two terms alters, affecting the solution accordingly.

Activity 8: Show that the following optimization problem has the same solution as the one in (8). We call these two optimization problems equivalent.

When looking at both equations (8) and (9), we can see that the constraint w_k is the same for both equations, now if we expand (8):

$$\frac{1}{2}\alpha w_k^T w_k + \frac{1}{2}(1-\alpha)(w_k - w_{k-1})^T(w_k - w_{k-1})$$

$$= \frac{1}{2}\alpha w_k^T w_k + \frac{1}{2}(1-\alpha) w_k^T w_k - \frac{1}{2}(1-\alpha) w_{k-1}^T w_k - \frac{1}{2}(1-\alpha) w_k^T w_{k-1} + \frac{1}{2}(1-\alpha) w_{k-1}^T w_{k-1}$$

Above we can see that when equation (8) is expanded it shows a combination of the terms that are outlined in equation (9). While it also has another set of terms with are labelled by w_{k-1} which also can be reorganized to take the same form as equation (9). Therefore, this proves that both equations (8) and (9) are equivalent and will exhibit the same solution.

Activity 9: Derive the IQP formulation in (20). Explain the rational for the additional constraints on the value of b.

To derive the IQP formulation for equation (20); which is used for the optimization problem for finding the parallel separating line barriers with the largest gap, we must first use the equation for the gap distance between the two lines:

$$d = d_r + d_l = \frac{2}{\sqrt{(w^T w)}}$$

$$d = \frac{2\sqrt{(w^T w)}}{w^T w}$$

$$\frac{\sqrt{(w^T w)}}{d} = \frac{w^T w}{2}$$

Now we must maximize d, to do so, we will minimize the right side of the equation,

$$\min_w \frac{1}{2} w^T w$$

When this formulation is derived, we also have some constraints that follow with this,

$$w^T p_i + b - 1 \geq 0$$

$$w^T p_j + b + 1 \leq 0$$

$$-1 + \epsilon \leq b \leq 1 - \epsilon$$

The additional constraints on the value of b are placed to set/define safe limits on the right and left of the autonomous vehicle. We also see another variable, ϵ , which is described to be a small number greater than 0, which acts as a buffer of sorts.

Activity 14: Plot the vehicle velocity vsas a function of $dmin$ and use it to explain what (28) is trying to accomplish. Identify the relevant sections in the code and the parameters in "params.yaml". Explain how different parameters in (28) can impact the vehicle commanded velocity, vs.

As the vehicle moves forward, dmin decreases as it nears an obstacle. Consequently, vs adjusts according to a prescribed formula to ensure appropriate deceleration, thereby averting collisions.

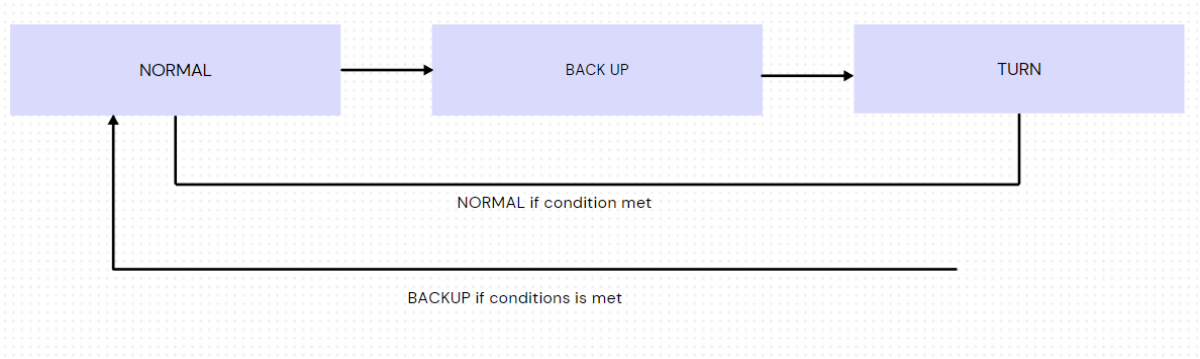
The graph features the dependent variables dstop, ddec, and vso, all defined in the params.yaml file. Specifically, dace (stop distance decay) is 1m, dstop (stop distance) is 0.6m, and vs0 (nominal speed) is 2m/s (as indicated in figure 14). Each parameter affects the system uniquely:

Vs0: This sets the vehicle's desired or nominal velocity when no obstacles are nearby, serving as the upper-speed limit. As dmin increases (indicating distance from obstacles), the vehicle's speed tends towards this nominal value, evident in the graph's trend towards two as dmin increases.

dstop: This determines the distance the vehicle must halt to avoid collision. As dmin approaches dstop, the maximum term between dmin-dstop and 0 becomes 0, causing vs to gradually decline to 0 until reaching dstop. This is observable in the graph as vs decreases to 0 as dmin approaches dstop (0.6m).

ddec: governs the rate of deceleration as the vehicle nears an obstacle. Higher values of dace result in faster deceleration when approaching an obstacle, transitioning from nominal speed to a stop more rapidly.

Activity 15: The “back-up and turn” functionality is implemented using a finite-state machine with three states: “normal”, “backup” and “turn”. Carefully examine the Python code in “navigation_lab9_inc.py” for the implementation of this functionality. Draw the finitestate machine diagram with all necessary details to explain the vehicle operation. You are encouraged to suggest (and implement) improvements to this functionality as you see fit.



The provided Python script implements a finite-state machine to enable a vehicle to perform backup and turn maneuvers when encountering obstacles. The finite-state machine comprises three states: "normal", "backup", and "turn".

- In the "normal" state, the vehicle operates as usual, navigating its environment.

- Upon detecting an obstacle too close to the vehicle, it transitions to the "backup" state. Here, it reverses its course for a specified duration or until conditions permit.

- Following the backup, the vehicle enters the "turn" state, executing a turn maneuver to circumvent the obstacle and resume regular operation.

This approach ensures adaptive and efficient navigation, particularly when obstacles obstruct the vehicle's path. To enhance the code further, consider:

- Parameterization for easy tuning of variables.

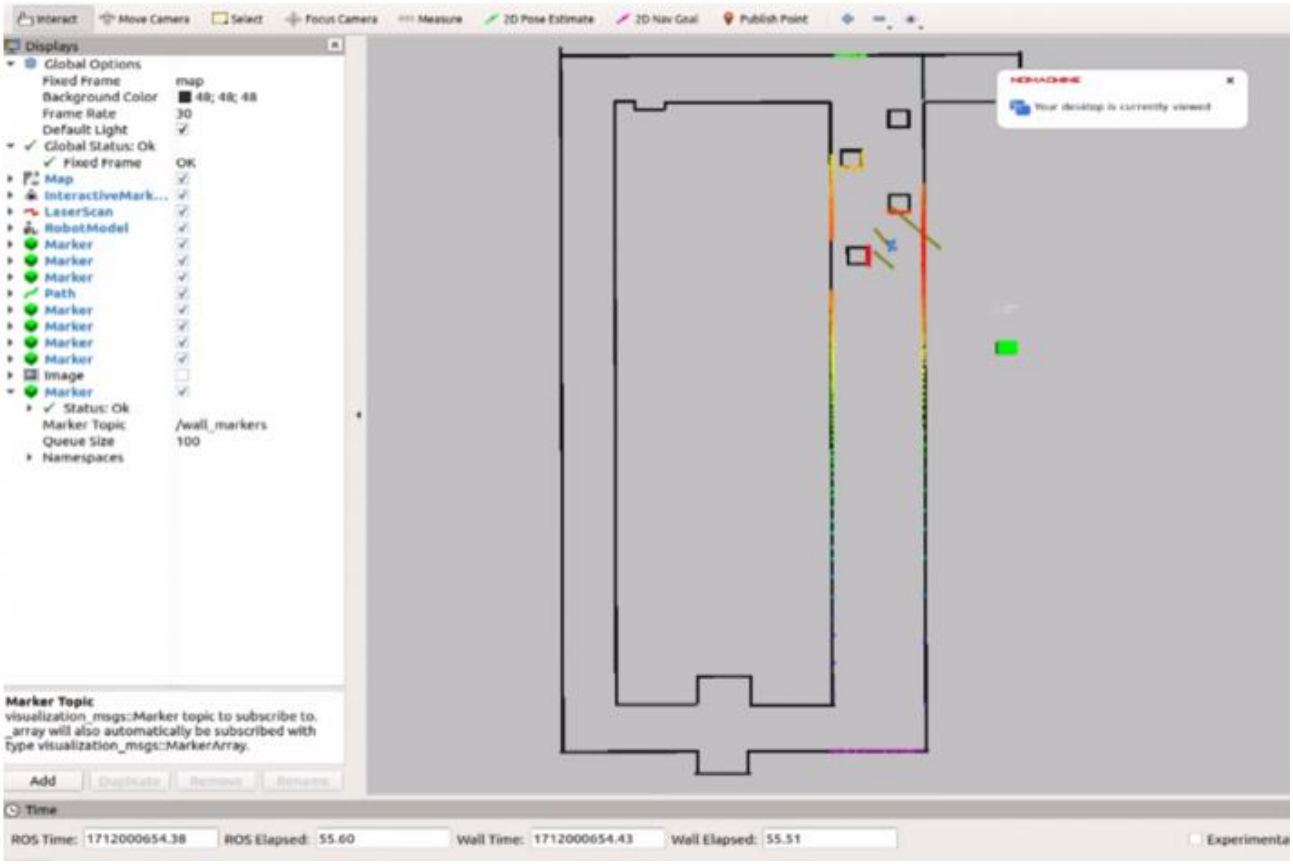
- Modularization to improve code organization and maintainability.

- Error handling to bolster robustness against unexpected scenarios.

- Optimization for enhanced performance.

- Thorough testing across various scenarios to ensure safety and reliability.

Activity 16: Launch the f1tenth_simulator "simulator.launch" file. Add a display of Marker type with the topic "wall_markers" to the "rviz" visualization environment. These markers represent the virtual separating line barriers and desired heading direction of the vehicle. Use the simulator environment to fine-tune the values of the control algorithm parameters until you achieve a satisfactory response in the simulation environment. You must try both formulations for finding the virtual separating line barriers in (9) and (2). Note that the value of the parameter "optim_mode" determines which formulation is used. Pay close attention to the "wall_markers" as you drive the vehicle around in the simulation environment.



Activity 17: Compare the performance of the self-driving algorithm using the two optimization formulations in (9) and (20). Briefly report your observations on the impact of the parameters "k_p", "k_d" and "n_pts_l", "n_pts_r", and "safe_distnace" on the vehicle self-driving performance.

```
158    n_pts_l: 100
159    n_pts_r: 100
160    tau: 0.1
161    k_d: 4.0
162    k_p: 3.5
```

k_p & k_d: Small differences of 1.0 in k_p and k_d can significantly affect the vehicle's response to errors and disturbances. Higher values of k_p may lead to a more aggressive response to errors, while higher k_d values can enhance stability but might introduce overshooting.

n_pts_l and n_pts_r: Increasing or decreasing the number of points in the left and right fields of view by 100 can impact the vehicle's ability to detect obstacles. More points provide a denser perception range but may increase the computational load.

Overall, even subtle variations in these parameters can significantly influence the self-driving algorithm's performance, highlighting the importance of parameter tuning to achieve desired outcomes regarding safety, efficiency, and responsiveness.

Activity 19: Adjust the controller parameters as you see necessary to achieve a satisfactory response. Briefly report on your observations from the experiments. Comment on any differences that you may observe between the system responses under the two optimization modes.

In our experiments, we adjusted the controller parameters to fine-tune the performance of the self-driving algorithm. We observed that small changes in parameters such as kp and kd significantly affected the vehicle's response to errors and disturbances. Increasing kp led to a more aggressive response to errors, while higher kd values improved stability but could introduce overshooting.

Furthermore, we experimented with the parameters nptsl and nptsr , which control the number of points in the left and right fields of view. Increasing these parameters resulted in a denser perception range, enhancing the vehicle's ability to detect obstacles. However, this also increased the computational load, which should be considered for real-time applications.

Overall, our observations highlight the importance of parameter tuning in optimizing the self-driving algorithm's performance. Careful adjustment of these parameters is crucial for achieving desired safety, efficiency, and responsiveness outcomes. We also noted that the algorithm's performance varied depending on the optimization mode used, emphasizing the need for thorough experimentation and testing to identify the most suitable parameter configurations.

Activity 20: Reflect on the operation of the self-driving control algorithm and suggest potential ways for improving it as you see fit.

Reflecting on the operation of the self-driving control algorithm, several potential areas for improvement and optimization emerge:

Sensory Perception Enhancement: Improving the sensors' accuracy and range can enhance the algorithm's ability to perceive the environment accurately. This could involve using higher-resolution cameras, more advanced LiDAR systems, or incorporating additional sensor modalities like radar or ultrasonic sensors.

Data Fusion and Processing: Implementing more sophisticated algorithms for data fusion and processing can help integrate information from different sensors more effectively. This could involve Kalman filtering, sensor fusion with deep learning models, or Bayesian inference methods.

Path Planning and Decision Making: Enhancing the algorithm's decision-making capabilities can lead to more efficient and safer driving behaviour. This could include developing more robust path-planning algorithms considering dynamic obstacles, traffic conditions, and complex road geometries.

Real-time Performance Optimization: Optimizing the algorithm's computational efficiency and real-time performance is crucial for deploying it in practical applications. This could involve implementing algorithmic optimizations, parallel processing techniques, or deploying the algorithm on specialized hardware platforms like GPUs or FPGAs.

Safety and Redundancy Measures: Incorporating redundant systems and safety mechanisms can enhance the algorithm's reliability and robustness. This could include implementing fail-safe modes, redundancy in sensor systems, and integrating safety-critical components like emergency braking systems.

Human-Machine Interaction: Designing intuitive and user-friendly interfaces for human-machine interaction can improve user acceptance and trust in autonomous driving systems. This could involve developing clear visualizations of the algorithm's decision-making process, providing feedback to the user, and ensuring transparent communication of system capabilities and limitations.

Addressing these areas for improvement can make the self-driving control algorithm more capable, reliable, and safe, paving the way for widespread adoption of autonomous driving technology.