EE 3EY4
Lab 5
-

Stefan Tosti - Tostis - 400367761 - L05
Luke Hendrikx - Hendrikl - 400388795 - L05
Soham Patel - Pates206 - 400380121 - L05

2024 - 02 - 23

Objective 2

This objective was led by Soham. Stefan was responsible for going over the lab manual and instructing the team what to do, and Luke was responsible for compiling all relevant information into the lab manual and answering questions.

Question 1

The q-axis current is aligned with the rotor's magnetic field in the rotating frame of the motor. The interaction between this q-axis current and the magnetic field generated by the permanent magnets in the motor is what produces torque in the motor. The magnitude of q-axis current is directly proportional to the amount of torque produced by the motor - in this way, it can be said that q-axis current and motor torque share a direct proportionality.

Question 2

When we use the apt-get command to install VESC, we are getting a pre-compiled version of the most up-to-date code for VESC. When we clone VESC from the 3EY4 repository, we are getting the actual source code of a VESC version that has been specifically designed and tested for 3EY4. The reason that we need to import the repository from Github is because we will need to change some of this source code in the lab, and we will not be able to do that with a pre-compiled version.

Question 3

Static_cast is used to convert a given expression to a certain type. In the code, static_cast is used to convert the expression within the brackets (I.e. payload_.first + 18) into unsigned 32 bit integers. We have to use static_cast in this code block because the data pointed to by payload_.first may not be an unsigned 32 bit integer. We convert the value to ensure that the bit shifting operation is performed on the desired data type.

Question 4

The code in Figure 1 is responsible for ensuring that we can receive q-axis current data from VESC by 17^{th}, 18^{th}, 19^{th} & 20^{th} bytes are transferred to the VESC. We add 17 to the value pointed to by payload_.first to access the 17th element in the payload_.first array. We then shift this value 24 bits left which puts us in the first byte position of a 32 bit unsigned integer. We repeat this same process for elements 18, 19 and 20, each time shifting 8 bits less. This means that we now have array elements 17, 18, 19 and 20, in the first, second, third and fourth byte positions respectively. Finally, we cast this value to a double, divide it by 100 and return it to the caller. This is important to parsing the q-current data because this data is stored in array elements 17, 18, 19, 20. As explained above, the bit shifting is required in order to store each element array at a specified position in the unsigned 32-bit integer.

Question 5

Declaring the current_qaxis() method in the vesc_packet.h file is what actually gives us access to the method as defined in the cpp file. We need to add function headers into the header file so that we can actually access the associated code and use the defined functions. This allows VESC to actually utilize the current_qaxis() method, which we had previously written to obtain q-axis current data by harnessing position 17, 18, 19 and 20 in the payload_.first array.

Question 6

The first part of this code is values->current_qaxis(). This is calling the current_qaxis() method on the values object. We use the -> operator here to access attributes of an object through a pointer. The second part of this code is state_msg ->state.current_q. This is responsible for assigning the value returned by values->current_qaxis() to the current_q attribute of the state_msg object.

Question 7

By doing this, we are including current_q as a member of the VescStateStemped sub-message. This allows VescStateStamped to be published successfully by the VESC node.

Question 8

In doing this, we are changing the directory at which the device stream is pulling its data from. In this code, we are changing the default data stream to the path defined by "/dev/sensors/vesc". Since the VESC system is a USB data stream with respect to ROS, we are essentially telling ROS to, by default, look at the provided directory for an incoming data stream.

Objective 3

This objective was led by Luke. Soham was responsible for going over the lab manual and instructing the team what to do, and Stefan was responsible for compiling all relevant information into the lab manual and answering questions.

Question 9

rostopic: This allows us to access ROS topics

pub: This is a sub-command of rostopic that allows us to publish a message to a topic.

-1: This is an option in the *pub* sub-command, which specifies that we should only publish one message before exiting

/commands/motor/speed: This is the name of the topic to which the message is published

std_msgs/Float64: This is the type of message that will be published. *std_msgs* is a package that includes message types in ROS. *Float64* is a message type used for 64-bit floating point numbers.

- -*:* This separates the topic type from the message to be published.

"7500.0": This is the message to be published to the topic

Objective 4

This objective was led by Stefan. Luke was responsible for going over the lab manual and instructing the team what to do, and Soham was responsible for compiling all relevant information into the lab manual and answering questions.

Question 10

Adding *catkin_install_python()* into *CMakeList.txt* file is important because it allows Python scripts to be executed as a part of our ROS package. Without adding this command, our Python scripts may not execute properly.

Question 11

Line 1 of this code is responsible for including the launch files for the F710 joyjoystick drivers. This allows the controller and its joysticks to interact with ROS. Similarly, line 2 is responsible for including the launch files for the VESC motor controller driver. This is what allows the motor controller to interface with ROS.

Line 3 creates a node named "JoyControl", which is a Python file called "JoyControl" which exists in the "test" package. Similar to this, Line 7 is responsible for creating a node named "servoControl" which is a Python file called "servControl", also existing in the "test" package.

Line 4 is responsible for setting the value of "Multiplier" within the "JoyControl" node to a value of "0.12". This "Multiplier" value is used to scale the input of the joysticks on the controller. Increasing the multiplier increases the sensitivity of the joystick, and vice versa for decreasing the multiplier value. Similarly, line 5 is responsible for setting the value of "ControlMethod" within the "JoyControl" node to a value of 0.

The value of "Multiplier", as mentioned above, is used to scale the input from the joysticks. A larger multiplier results in a more sensitive joystick input, and a smaller multiplier results in a less sensitive joystick input. We know that the speed of the motor is controlled by the up and down movements of our joystick, and is also dictated by the duty cycle of the incoming signal (direct proportionality) meaning that the multiplier value has a direct correlation to the duty cycle. Increasing the multiplier means that each unit of the joystick input will result in a larger

change of the joystick duty cycle. The opposite will be true for a decreasing multiplier. The "controlMethod" specifies the type of control (e.g., duty cycle, speed, current) applied to the AEV, altering its response to input commands. For instance, selecting the duty cycle control method changes how the AEV accelerates and decelerates based on the joystick's input.

Question 12

The line a = -1*data.axes[0]*0.5+0.5 scales with the line in the callback function of servoControl.py and it changes the direction of the servo motor. This line of code changes the servo's position which depends on the joystick according to the line. The joystick's horizontal axis input is inversely scaled and translated, which effectively maps it to the range to control the servo angle, with a value of zero corresponding to the center position of the servo. This is how the servo is manipulated so that moving the joystick left or right steers the vehicle to the left or right.

Objective 5

This objective was led by the team. Everyone contributed to this objective equally.

Question 13

The screenshot below shows our data output from driving the car. As can be seen, we have plotted speed, as well as q-axis current versus time on two separate axes. Visually, we can definitely see a direct proportionality between speed and q-axis current, that is to say, when speed is increasing and positive, q-axis current is also increasing and positive, and vice versa. Looking at just past t = 2 seconds, we begin to accelerate the car forwards - we see the speed increase sharply, as well as a large in-flux of q-axis current. Similarly, if we look at t = 4 seconds, we can see that the car has come to a sudden stop. The speed shoots down to 0 very quickly, and then settles at 0. Comparing this to the q-axis current, we see a sharp, negative dip right at t = 4s, and then a settling current of 0. From our class lectures, we know that there is a direct mathematical relationship between q-axis current and speed, which can be seen through torque as an intermediate variable. Referring to the Permanent Magnet Flux Linkage lecture series on Avenue, we know that the q-axis current is called the "torque producing current ". Torque production occurs when phase current is in phase with phase voltage, which creates positive air gap power, and hence positive torque. Maximum torque occurs when all of the current is present on the q-axis. We also know, from the same lecture series, that…

$$\varepsilon_q = \lambda_d \omega_e = \lambda_d \left[\left(\omega_{mech} \right) (\#PP) \right] = \lambda_d \left[(RPM) \left(\frac{\pi}{30} \right) (\#PP) \right]$$

Isolating for RPM in the above equation…

$$RPM = \left[\frac{\left(\frac{\varepsilon_q}{\lambda_d} \right)}{\#PP} \right] \left[\frac{30}{\pi} \right]$$

We can thus see that RPM is directly proportional to q-axis voltage, and q-axis voltage is directly proportional q-axis current. This means that the higher the q-axis current, the higher the q-axis voltage, the higher the torque, the higher the RPM.

This relationship can be seen even easier when referring to the subsequent lecture module, Torque Production. In this module, we derive the following relationship…

$$T = \left(\frac{3}{2}\right)(\#PP)\left(\lambda_{PM}\right)\left(i_q\right)$$

Which very clearly shows that Torque and Q-axis current are directly and proportionally related to each other. We can then, by the same logic as used above, argue that torque is directly proportional to RPM (speed).