ROS2 Intermediate Introduction

1 Introduction

The Robotic Operating System (ROS) 2 is a set of software libraries and tools for building robot applications. There are two primary distributions of the ROS, known as ROS and ROS2. The differences between these distributions are irrelevant for this training beyond knowing which version we are using. Previous training techniques on ROS2 have neglected one of the most interesting components of robotics programming: controlling motors and other physical components. This guide is intended to introduce students to writing code to control motors and to creating nodes intended for a robotics application. This training covers the following concepts:

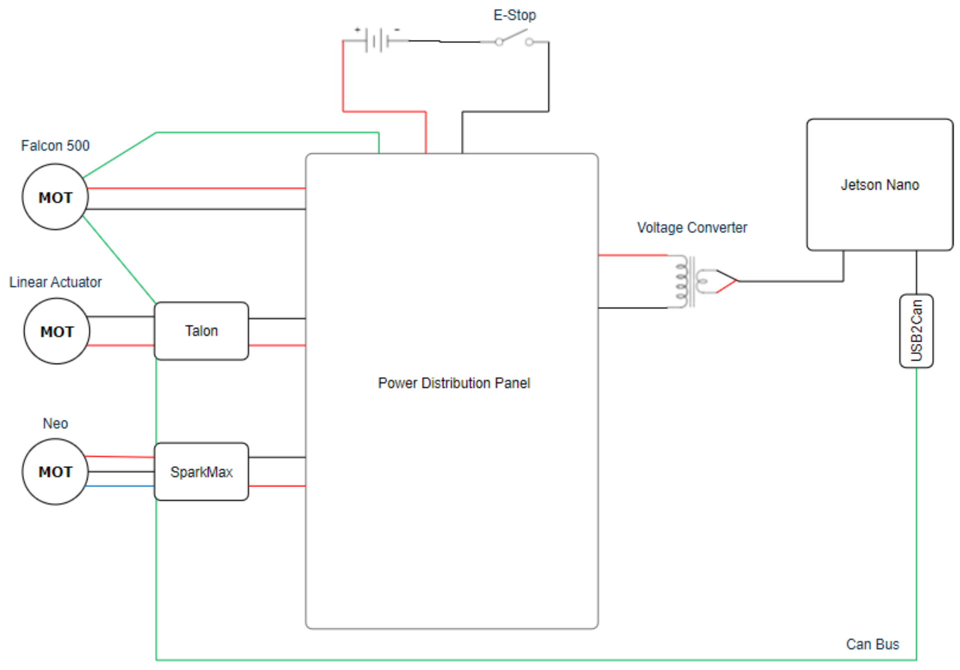
- Creating ROS2 subscribers
- Compiling code using colcon
- Understanding existing ROS2 codebase
- Writing ROS2 Subscriber Callback Functions
- Creating ROS2 node files

1.1 Prerequisites

Students should be familiar with several concepts, including ROS2 concepts, Linux, VirtualBox virtual machines, and using the command line.

2 Electrical Circuit

The training board has a circuit that has three motors attached. The completed circuit is shown in Figure 1, shown below. This circuit is a good representation of the robot, as it contains all the motor controllers and motors currently on the robot. The motor controllers attached are a Vex Talon SRX, a Vex Talon FX and a Rev SparkMax. The attached motors are a Vex Falcon 500, a linear actuator, and a Rev Neo brushless motor. The battery or power converter is attached to the circuit and powers the power distribution panel (PDP). The regulates the output voltage and amperage flowing to the devices that are attached. The Jetson Nano controls the various motors and motors controllers using the Controller Area Network (CAN) bus. The Jetson also communicates with the client and receives the commands sent from the client.



	DRAWN BY	DATE	VERSION	PAGE NO.
WIRING DIAGRAM RAZORBOTZ WIRING TRAINING	Andrew Burroughs	09/28/2023	1.0	1

Figure 1: Wiring Diagram

3 Existing Nodes

The node structure for this project are shown in Figure 2. The communication node and the logic nodes are given. The launch file is also supplied. The communication node handles the communication between the client GUI and the Jetson. This node receives the various commands from the client and sends the values to the Logic node via the topic names shown. The Logic node takes these values and publishes them with the names talon_speed, falcon_speed, and neo_speed. The Talon, Falcon, and Neo nodes are missing the main C++ file, but have the CMakeLists.txt and package.xml files.

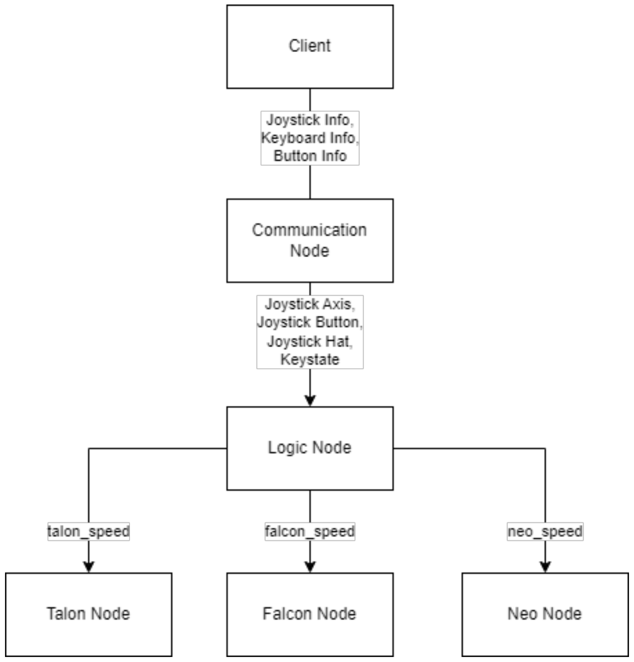


Figure 2: Existing Nodes

4 Tasks

To complete the tasks in this file, the student needs to create the Talon, Falcon, and Neo nodes. The student will need to create the C++ files for each node and the contents of each file to allow the motors to run.

5 Uploading the Code

After completing and compiling the code, create a new branch on the Razorbotz Github Training page at https://github.com/Razorbotz/Training. Upload the new code to this branch. Log on to the Jetson Nano and navigate to the Training folder. Download the new branch and switch to it using the git switch command. Compile the code with colcon and check it for any compilation errors. If the code has compiled correctly, launch the program using ros2 launch. Using the Razorbotz with the GUI client, connect to the robot. If the code was written correctly, the motors should move when the user moves the joystick attached to the laptop. If an error occurs, talk to the Razorbotz Computer Systems team lead.